

# Quantum Computing

Nihar Shah

August 26, 2024

## Quantum Computing: Theory and Practice

Nihar Shah

Guide: Professor Phani Motammari



*Department of Computational and Data Science  
Indian Institute of Science*

## Abstract

The following content consists of the work done by me as a part of my Master's Thesis at Indian Institute of Science, Bangalore. The work is done under the guidance of Prof. Phani Motammari. This work are my notes made during the 1 year work from June 2024 to August 2025 and are based on the lectures, papers, books and other resources that I have referred to during the course of my work. To a reader who is not at all familiar with Quantum Computing, this work will serve as a good starting point. I suggest him to start reading from the Appendix to get a good grasp on the maths used in the entire text. The work is divided into 3 parts: The first part is the introduction to Quantum Computing, the second part is the Advanced Quantum Computing and the third part is the work done by me as a part of my thesis and the Appendix. Along with the references, I have also included the code snippets that I have written during the course of my work. The references are included at the end of the document. I hope that this text helps the reader to get a good grasp on Quantum Computing. Enjoy reading!

## Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor, Professor Phani Motammari, for the continuous support of my study and related, for his patience, and immense knowledge. His guidance helped me through the time of research and writing of this thesis.

Besides my advisor I am also grateful to the faculty members and staff at the Department of Computational and Data Science of Indian Institute of Science for their assistance and support. Special thanks to MATRIX lab for providing a stimulating and collaborative research environment.

Last but not the least, I would like to thank my family: my parents, for supporting me spiritually throughout writing this thesis and my life in general.

# Contents

<b>Acknowledgements</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Why Quantum Computation? . . . . .	1
1.2 The Double Slit Experiment . . . . .	2
1.3 Computation . . . . .	6
1.4 The coin Experiment: Analogy for intuition . . . . .	8
1.5 Quantum Information . . . . .	12
1.5.1 A probabilistic model . . . . .	12
1.5.2 Quantum bits (qubits) . . . . .	14
1.5.3 Example of Qubits . . . . .	20
1.5.4 Multiple Qubits . . . . .	21
1.6 Measurements . . . . .	33
1.6.1 Simple Measurements . . . . .	33
1.6.2 Measuring Multiple Systems . . . . .	41
<b>2 Axioms of Quantum Mechanics</b>	<b>46</b>
2.1 Postulate 1: The State Space Postulate . . . . .	46
2.2 Postulate 2: Evolution Postulate . . . . .	48
2.3 Postulate 3: Measurement Postulate . . . . .	49
2.3.1 Projective Measurements . . . . .	52
2.3.2 Positive Operator Valued Measure (POVM) measurements . .	55
2.4 Postulate 4: Composite Systems . . . . .	56
2.4.1 Full and Partial Measurements of a multi-particle systems . .	58
<b>3 Qubits</b>	<b>60</b>
3.1 Bloch-Poincare sphere representation of a Qubit . . . . .	60
3.2 Single Qubit Gates . . . . .	65
3.2.1 Pauli Matrices . . . . .	65

3.2.2	Hadamard Gate . . . . .	73
3.2.3	$S$ and $S^\dagger$ Gate . . . . .	76
3.2.4	$T$ and $T^\dagger$ Gate . . . . .	79
3.2.5	Quantum Rotation Gates . . . . .	82
3.2.6	Universal Quantum Gates . . . . .	85
3.3	Multi-Qubit Gates . . . . .	86
3.3.1	Two-qubit Gates . . . . .	86
3.3.2	Three-qubit Gates . . . . .	96
3.4	Quantum Circuits . . . . .	104
<b>4</b>	<b>Entanglement</b>	<b>111</b>
4.1	Bell States/EPR Pairs . . . . .	112
4.2	No Cloning Theorem . . . . .	115
4.3	Applications of Entanglement . . . . .	116
4.3.1	Quantum Teleportation . . . . .	116
4.3.2	Superdense Coding . . . . .	120
<b>5</b>	<b>Quantum Algorithms</b>	<b>125</b>
5.1	Classical Circuits simulating Quantum Circuits . . . . .	126
5.2	Quantum Computation . . . . .	127
5.3	Linear Growth and Duhamel's Principle . . . . .	127
5.4	Universality of various sets of elementary gates . . . . .	129
5.5	Universal gate sets and Reversible Computation . . . . .	130
5.6	Quantum Circuits simulating Classical Circuits . . . . .	133
5.7	Computational Complexity Classes . . . . .	136
5.8	Quantum Algorithms as Quantum Circuits . . . . .	137
5.9	Fixed Point Number Representation and Classical Arithmetic operations	138
5.10	Fault tolerant Computation . . . . .	139
5.11	Complexity of Quantum Algorithms . . . . .	139
5.12	Oracles . . . . .	141
5.13	Quantum Parallelism . . . . .	146
<b>6</b>	<b>Elitzur-Vaidman Bomb Detection Algorithm</b>	<b>148</b>
6.1	The Problem . . . . .	148
<b>7</b>	<b>Deutsch's Algorithm</b>	<b>154</b>
7.1	The Problem Definition . . . . .	154
7.2	The Classical Solution . . . . .	155
7.3	The Quantum Solution: Deutsch's Algorithm . . . . .	156

7.3.1	Idea of the Algorithm . . . . .	158
7.3.2	Algorithm . . . . .	159
7.3.3	Example . . . . .	162
7.4	Implementation using Qiskit . . . . .	165
7.4.1	Background . . . . .	165
7.4.2	Requirements . . . . .	165
7.4.3	Setup . . . . .	166
<b>8</b>	<b>Deutsch-Jozsa Algorithm</b>	<b>173</b>
8.1	The Problem Definition . . . . .	173
8.2	The Classical Solution . . . . .	174
8.3	The Quantum Solution: Deutsch-Jozsa Algorithm . . . . .	175
8.3.1	Idea of the Algorithm . . . . .	175
8.3.2	Algorithm . . . . .	175
8.3.3	Example . . . . .	179
8.4	Complexity Analysis . . . . .	182
8.5	Conclusion . . . . .	183
8.6	Implementation using Qiskit . . . . .	183
8.6.1	Background . . . . .	183
8.6.2	Requirements . . . . .	183
8.6.3	Setup . . . . .	183
<b>9</b>	<b>Simon’s Algorithm</b>	<b>194</b>
9.1	The Problem Definition . . . . .	194
9.2	The Classical Solution . . . . .	195
9.3	The Quantum Solution: Simon’s Algorithm . . . . .	196
9.3.1	Algorithm . . . . .	196
<b>10</b>	<b>Bernstein-Vazirani Algorithm</b>	<b>202</b>
<b>11</b>	<b>Grover’s Search Algorithm</b>	<b>203</b>
11.1	The Problem Definition . . . . .	203
11.2	The Classical Solution . . . . .	204
11.3	The Quantum Solution: Grover’s Algorithm . . . . .	205
11.3.1	Idea of the Algorithm . . . . .	205
11.3.2	Algorithm . . . . .	209
11.3.3	Example . . . . .	218
11.4	Complexity Analysis . . . . .	222
11.4.1	Unique Search Problem . . . . .	223

11.4.2 Multiple Solutions . . . . .	225
11.4.3 Trivial Case . . . . .	227
11.5 Conclusion . . . . .	228
11.6 Amplitude Amplification . . . . .	229
11.7 Applications of Grover's Algorithm . . . . .	229
11.7.1 Application: Satisfiability . . . . .	229
11.8 Implementation using Qiskit . . . . .	229
11.8.1 Background . . . . .	229
11.8.2 Requirements . . . . .	229
11.8.3 Setup . . . . .	229
<b>12 Phase Estimation</b>	<b>236</b>
12.1 Phase Estimation . . . . .	236
12.1.1 Classical Method . . . . .	237
12.1.2 Hadamard Test . . . . .	238
12.1.3 Kitaev's Method - for Quantum Phase Estimation . . . . .	242
12.1.4 Phase Estimation Procedure . . . . .	244
12.1.5 Quantum Fourier Transform . . . . .	254
<b>13 Shor's Algorithm</b>	<b>258</b>
<b>14 The HHL Algorithm</b>	<b>259</b>
14.1 Introduction . . . . .	259
14.2 The Linear-System Problem (LSP) . . . . .	259
14.3 The HHL Algorithm . . . . .	261
14.3.1 Some Mathematical Preliminaries . . . . .	261
14.3.2 Algorithm . . . . .	263
14.3.3 Example . . . . .	270
14.4 Complexity Analysis . . . . .	274
14.5 Improvements in HHL algorithm . . . . .	278
14.6 Implementation using Qiskit . . . . .	278
14.7 Applications . . . . .	278
14.7.1 Poisson's problem . . . . .	278
14.7.2 Solve Linear Differential Equations . . . . .	281
<b>15 Trotter based Hamiltonian Simulation</b>	<b>287</b>
15.1 The Problem Definition . . . . .	287
15.2 Trotter Splitting . . . . .	288
15.3 Commutator type error bound . . . . .	289

<b>16 Block Encoding</b>	<b>292</b>
16.1 Query model for matrix entries . . . . .	292
16.2 Block Encoding . . . . .	294
16.3 Block Encoding of s-sparse matrix . . . . .	299
16.4 Hermitian Block Encoding . . . . .	301
16.5 Query models for general sparse matrices . . . . .	301
<b>17 Quantum Subspace Methods</b>	<b>305</b>
17.1 Basic Quantum Computing Concepts . . . . .	306
17.2 Single-qubit gates . . . . .	306
17.3 Two-qubit gates . . . . .	307
17.4 Classical Subspace Methods . . . . .	308
17.4.1 Krylov Space Methods . . . . .	311
17.4.2 Lanczos Method . . . . .	311
17.5 Quantum Subspace Methods . . . . .	311
17.5.1 Idea of the Algorithm . . . . .	311
17.5.2 Difference between QSE and VQE . . . . .	311
17.5.3 Hamiltonian Krylov Space Methods . . . . .	312
<b>18 Variational Algorithms</b>	<b>313</b>
18.1 Variational Theorem . . . . .	316
18.1.1 Mathematical intuition for energy and ground state . . . . .	316
18.1.2 Variational theorem of Quantum Mechanics . . . . .	317
18.2 Reference States . . . . .	318
18.2.1 Default State . . . . .	318
18.2.2 Classical Reference state . . . . .	318
18.2.3 Quantum Reference State . . . . .	319
18.2.4 Constructing Reference States using template circuits . . . . .	320
18.2.5 Application Specific Reference States . . . . .	321
18.3 Ansatze and Variational Forms . . . . .	322
18.3.1 Parameterized Quantum Circuits . . . . .	322
18.3.2 Variational Form and Ansatz . . . . .	324
18.3.3 Problem-specific ansatze . . . . .	328
18.4 Cost Functions . . . . .	331
18.4.1 Primitive . . . . .	332

<b>A Linear Algebra and Calculus Prerequisites</b>	<b>336</b>
A.1 Linear Algebra . . . . .	337
A.1.1 Mathematical Notation . . . . .	337
A.1.2 Basics of Linear Algebra . . . . .	337
A.1.3 Inner Product . . . . .	343
A.1.4 Gram-Schmidt Orthogonalization . . . . .	347
A.1.5 Cauchy-Schwarz and Triangle Inequalities . . . . .	349
A.1.6 Linear Operators and Matrix Representation . . . . .	350
A.1.7 Hermitian and Unitary Operators . . . . .	356
A.1.8 Eigen Value Problems . . . . .	361
A.1.9 Norms . . . . .	372
A.1.10 Induce Matrix Norm . . . . .	373
A.1.11 Conditioning and Condition Number . . . . .	374
A.1.12 Overview of Iterative Methods . . . . .	376
A.1.13 Conclusions . . . . .	377
A.2 Tensor Products . . . . .	378
A.3 Complex Numbers . . . . .	382
A.3.1 Complex Plane . . . . .	382
A.3.2 Addition of Complex Numbers . . . . .	382
A.3.3 Multiplication of Complex Numbers . . . . .	383
A.3.4 Complex Conjugate . . . . .	384
A.3.5 Modulus of a Complex Number . . . . .	384
A.3.6 Polar form . . . . .	385
A.3.7 Euler's Formula . . . . .	385
A.3.8 Roots of Unity . . . . .	386
A.3.9 Discrete Fourier Transform Matrix . . . . .	387
A.3.10 Classical Fast Fourier Transform . . . . .	396
A.4 Probability Theory . . . . .	398
A.5 Calculus . . . . .	398
<b>B Classical Computations</b>	<b>399</b>
B.1 Classical Logic Gates . . . . .	400
B.1.1 NOT Gate . . . . .	400
B.1.2 AND Gate . . . . .	400
B.1.3 OR Gate . . . . .	401
B.1.4 NAND Gate . . . . .	401
B.1.5 NOR Gate . . . . .	402
B.1.6 XOR Gate . . . . .	403

B.1.7 FANOUT Gate . . . . .	404
B.1.8 CROSSOVER Gate . . . . .	404
B.2 Classical Circuits . . . . .	405
B.3 Classical Algorithms as logic circuits . . . . .	407
B.3.1 Multiplication problem and factoring problem . . . . .	407
B.4 Classical Circuits . . . . .	408
<b>C Qiskit</b>	<b>410</b>
C.1 Introduction to Qiskit . . . . .	410
C.1.1 The Qiskit SDK . . . . .	410
C.1.2 Qiskit Runtime . . . . .	411
C.1.3 Qiskit Serverless . . . . .	412
C.1.4 Qiskit Transpiler as a Service . . . . .	413
C.1.5 The Qiskit ecosystem . . . . .	413
C.2 Install Qiskit . . . . .	415
C.3 Anatomy of a Quantum Computing Service . . . . .	415
C.3.1 Getting set up with Qiskit . . . . .	417
<b>Bibliography</b>	<b>418</b>

# Chapter 1

## Introduction

### 1.1 Why Quantum Computation?

Quantum computers are the only model of computation that escape the limitations on computation imposed by the extended Church-Turing Thesis. These theoretical limitations will soon have practical consequences as Moore's Law (the doubling every eighteen months in transistor density on chips and the resulting increase in computer speed) is expected in a decade or so to run into inherent (classical) physical limits such as transistors scaling down to the size of elementary particles.

Quantum computers would have a profound effect on complexity based cryptography, via the polynomial time quantum algorithms for factoring and discrete logs. Understanding how to meet these threats by designing quantum resistant crypto systems or by restoring to quantum cryptography is a major challenge.

Quantum computation has brought a renaissance in the examination of the foundations of quantum mechanics: highlighting phenomena such as entanglement and the resulting exponential power inherent in quantum physics, and showing that quantum error-correcting codes may be used to bring stability into the seemingly inherent unstable quantum world.

Lastly, the study of quantum computation has helped broaden and deepen our insights into complexity theory (and tools such as Fourier Analysis and information theory), and in a few cases helped resolve open questions in classical complexity theory.

Nature at the sub-atomic scale behaves totally differently from anything that our experience with the physical world prepares us for. Quantum mechanics is the name of the branch of physics that governs the world of elementary particles such as electrons and photons, and it is paradoxical, unintuitive, and radically strange.

below is a sampling of a few such odd features.

- Complete knowledge of a system's state is forbidden - A measurement reveals only a small amount of information about the quantum state of the system.
- The act of measuring a particle fundamentally disturbs its state.
- Quantum entities do not have trajectories. All that we can say for an elementary particle is that it started at A and was measured later at B. We cannot say anything about the trajectory or path it took from A to B.
- Quantum mechanics is inherently probabilistic. If we prepare two elementary particles in identical states and measure them, the result may be different for each particle.
- Quantum entities behave in some ways like particles and in others like waves, but they really behave in their own unique way, neither particles nor waves.

These features are truly strange, and difficult to accept. To quote the great physicist Niels Bohr, "Anyone who is not shocked by quantum theory has not understood it". We start by describing a simple experiment that highlights many differences between quantum mechanics and our classical intuition. It is the famous double slit experiment. The intuition gained in the process will help us define qubits, and more generally in the study of quantum computing.

## 1.2 The Double Slit Experiment

What is the nature of light? You may have learned light is electromagnetic waves propagating through space. Also, you may have learned that light is made of a rain of individual particles called photon. but these two notions seem contradictory, how can it be both?

The debate over the nature of light goes deep into the history of science. The eminent physicist Isaac Newton believed that light was a rain of particles, called corpuscles. At the beginning of the nineteenth century, Thomas Young demonstrated with his famous double-slit interference experiment that light propagates as electromagnetic waves, and the debate seemed to be over. However, in 1905, Einstein was able to explain the photoelectric effect by using the idea of light quanta, or particles which we now call photons.

Similar confusion reigned over the nature of electrons, which behaved like particles, but then it was discovered in electron diffraction experiments, performed in

1927, that they exhibit wave behavior. So do the electrons behave like particles or waves? And what about photons? This great challenge was resolved with the discovery of the equations of quantum mechanics. but the theory is not intuitive, and its description of matter is very different from our common experience.

To understand what seems to be a paradox, we look to Young's double slit experiment. Here's the setup: a source of light is shone at a screen with two very thin, identical slits cut into it. Some light passes through the two slits and lands upon a subsequent screen. Take a look at the figure for a diagram of the experiment setup.

First, think what would happen to a stream of bullets going through this double slit experiment. The source, which we think of as a machine gun, is unsteady and sprays the bullets in the general direction of the two slits. Some bullets pass through one slit, some pass through the other slit, and others don't make it through one slits. The bullets that do go through the slits then land on the observing screen behind them. Now suppose we closed slit 2. Then the bullets can only go through slit 1 and land in a small spread behind slit 1. If we graphed the number of times a bullet that went through slit 1 and landed at the position  $y$  on the observation screen, we would see a normal distribution centered directly behind slit 1. That is, most land directly behind the slit, but some stray off a little due to the small amount randomness inherent in the gun, and because of they ricochet off the edges of the slit. If we now close slit 1 and open slit 2, we could see a normal distribution centered at slit 2.

Now let's repeat the experiment with both slits open. If we graph the number of times a bullet that went through either slit landed at the position  $y$ , we should see the sum of the graph we made for slit 1 and the graph for slit 2.

Another way we can think of the graphs we made is as graphs of the probability that a bullet will land at a particular spot on  $y$  on the screen. Let  $P_1(y)$  denote the probability that the bullet lands at point  $y$  when only slit 1 is open, and similarly for  $P_2(y)$ . And let  $P_{12}(y)$  denote the probability that the bullet lands at point  $y$  when both slits are open. Then  $P_{12}(y) = P_1(y) + P_2(y)$ .

Next, we consider the situation for waves, for example water waves. A water wave doesn't go through either slit 1 or slit 2, it goes through both. you should imagine the rest of 1 water wave as it approaches the slit. As it hits the slit, the wave is blocked at all places but the two slits, and waves on the other side are generated at each slit as depicted in the figure 1.1.

When the new waves generated at each slit run into each other, interference occurs. We can see this by plotting the intensity (that is, the amount of energy carried by the waves) at each point  $y$  along the viewing screen. What we see is the familiar interference pattern seen in figure 1.1. The dark patches of the interference

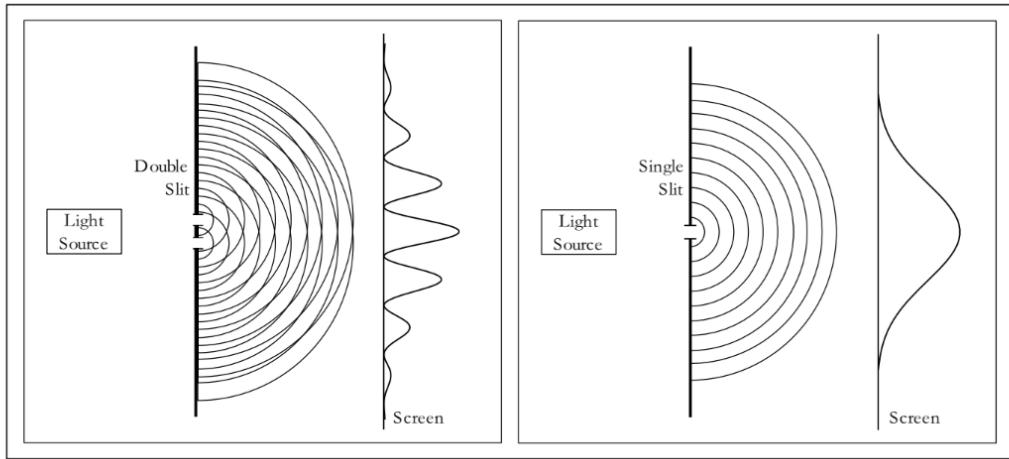


Figure 1.1: Double slit and single-slit diffraction

pattern seen in figure 1.1 occur where the wave from the first slit arrives perfectly out of sync with wave from the second slit, while the bright points are where the two arrive in sync. For example, the bright spot right in the middle is bright because each wave travels the exact same distance from their respective slits to the screen, so they arrive in sync. The first dark spots are where the wave from one slit traveled exactly half of a wavelength longer than the other wave, thus they arrive at opposite points in their cycle and cancel. Here, it is not the intensities coming from each slit that add, but the height of the wave. This differs in the case of bullets:  $I_{12}(y) \neq I_1(y) + I_2(y)$ , but  $h_{12}(y) = h_1(y) + h_2(y)$ , and  $I_{12}(y) = h(y)^2$ , where  $h(y)$  is the height of the wave and  $I(y)$  is the intensity, or energy, of the wave.

Before we can say what light does, we need one more crucial piece of information. What happens when we turn down the intensity in both of these examples?

In the case of bullets, turning down the intensity means turning down the rate at which the bullets are fired. When we turn down the intensity, each time a bullet hits the screen it transfers the same amount of energy, but the frequency at which bullets hit the screen becomes less.

With water waves, turning down the intensity means making the wave amplitudes smaller. Each time a wave hits the screen it transfer less energy, but the frequency of the waves hitting the screen is unchanged.

Now what happens when we do this experiment with light. As Young observed in 1802, light makes an interference pattern on the screen. From this observation he concluded that the nature of light is wavelike, and reasonably so!. However, Young

was unable at the time to turn down the intensity of light to see the problem with the wave explanation.

Picture now that the observation screen is made of thousands of tiny little photo-detectors that can detect the energy they absorb. For high intensities the photo-detectors individually are picking up a lot of energy, and when we plot the intensity against the position  $y$  along the screen we see the same interference pattern as described earlier. Now, turn the intensity of the light very very low. At first, the intensity scales down lower and lower everywhere, just like a wave. But as soon as we get low enough, the energy that the photo-detectors report reaches a minimum energy, and all of the detectors are reporting the same energy, call it  $E_0$ , just at different rates. This energy corresponds to the energy carried by an individual photon, and at this stage we see what is called quantization of light. Photo-detectors that are in the bright spots of the interference pattern report the energy  $E_0$  very frequently, while the darker areas report the energy  $E_0$  at lower rates. Totally dark points still report nothing. This behavior is the behavior of bullets not waves! We now see that photons behave unlike either bullets or waves, but like something entirely different.

Turn down, the intensity so low that only one photo-detector reports something each second. in other words, the source only sends one photon at at time. Each time a detector receives a photon, we record where on the array it landed and plot it on a graph. The distribution we draw will reflect the probability that a single photon will land at a particular point.

Logically we think that the photon will either go through one slit or the other. Then, like the bullets, the probability that the photon lands at a point  $y$  should be  $P_{12}(y) = P_1(y) + P_2(y)$  and the distribution we expect to see is the two peaked distribution of the bullets. But this is not what we see at all.

What we actually see is the same interference pattern as before! But how can this be? For there to be an interference pattern, light coming from one slit must interfere with light from the other slit; but there is only one photon going through at a time! The modern explanation is that the photon actually goes through both the slits at the same time, and interferes with itself. The mathematics is analogous to that in the case of water waves. We say that the probability  $P(y)$  that a photon is detected at  $y$  is proportional to the square of some quantity  $a(y)$ , which we call a probability amplitude. Now probability amplitudes for different alternatives add up. So  $a_{12}(y) = a_1(y) + a_2(y)$ . But  $P_{12}(y) = |a_{12}(y)|^2 \neq |a_1(y)|^2 + |a_2(y)|^2 = P_1(y) + P_2(y)$ .

Logically, we can ask which slit the photon went through, and try to measure it. Thus, we might construct a double slit experiment where we put a photo-detector at each slit, so that each time a photon comes through the experiment we see which slit it went through and where it hits on the screen. but when such an experiment

is performed, the interference pattern gets completely washed out! The very fact that we know which slit the photon goes through makes the interference pattern go away. This is the first example we see of how measuring a quantum system alters the system.

Here the photon looks both like a particle, a discreet package , and a wave that can have interference. It seems that the photon acts like both a wave and a particle, but at the same time it doesn't exactly behave like either. This is what is commonly known as the wave-particle duality, usually though of as a paradox. The resolutions is that the quantum mechanical behavior of matter is unique, something entirely new.

What may be more mind blowing still is that if we conduct the exact same experiment with electrons instead of light, we get the exact same results! Although it is common to imagine electrons as tiny little charged spheres, they are actually quantum entities, neither wave nor particle but understood by their wave function.

The truth is that there is no paradox, just an absence of intuition for quantum entities. Why should they be intuitive? Things on our scale do not behave like wave functions, and unless we conduct wild experiments like this we do not see the effects of quantum mechanics.

## 1.3 Computation

A computation corresponds to some input to a processor which then performs instructions may be using look-up tables(oracles)/libraries on input and produces an output. Similar is seen in the physical process in nature where for Physical processes in some initial state through some interactions goes to the final state. Note that not every physical process is a computation but for every computation there must be an underlying physical process.

Mathematical description of what goes on in a computer needs physical hardware to implement the task. Whether certain abstract mathematical operations are possible or not ultimately depends on whether there exists some suitable physical hardware to implement it. Thus, potentialities and limitation of a computer depends on the available hardware.

At the fundamental level there are only four types of interactions: Strong and weak interactions do not play any role in our day to day devices. Gravity acts as a fixed background but does not impact much of our devices. Electromagnetism is the mostly used force in its various forms to control and manipulate the dynamics of our devices. Tasks which can be performed on a computer are limited by the manner in

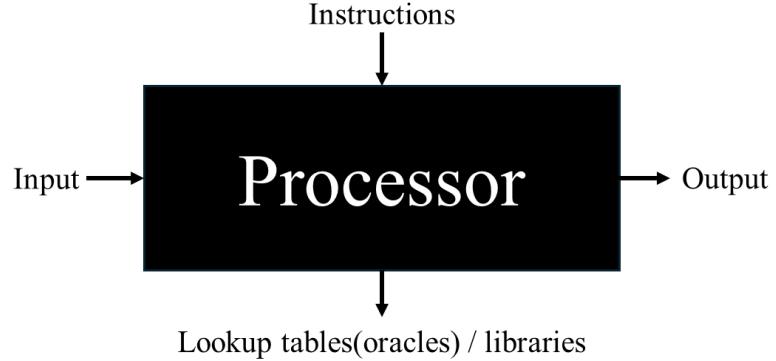


Figure 1.2: Computation

which we can use various aspects of electromagnetic interactions.

Computer is a device that purposefully processes meaningful information. From a philosophical point of view in order to understand a definition we need to keep on understanding the words in its definition till its intuition is rooted in our past experience of the physical world. Here, information means an abstract quantity by represented by some mathematical variables. Processes refers to the manipulation of physical states by a device. Meaningful relates to the mapping between the mathematical variables and physical state. Purposefully means contained in algorithmic design of instructions. For example, Boolean logic based computers have mathematical variables as  $\{0, 1\}$ . Physical states are  $\{\text{off}, \text{on}\}$  of transistors. Here 0 is mapped to off and 1 is mapped to on state. Algorithmic instructions manipulate the values 0 and 1.

It is a wonderful property of nature that physical properties and observations can be represented by abstract mathematical variables by designing a suitable map, and working on the variables without any reference to their physical origin and afterwards carrying out transformations insert physical meaning back. For example, a phone can be copied and then sent and then interpreted back as phone number. Thus, Physical data is converted into abstract mathematical information removing all its features of physical realisation which is then processed by rules of mathematics acting in terms of language and its grammar. Then its physical interpretation is practically useful knowledge. Note that data without its physical realisation is information and knowledge is processed information along with its physical interpretation. Thus, learning mathematical concepts needs a support of their physical realisations in some or the other form.

Now we can define what we want a computer to do. We want a general infor-

mation Processing system (which is limited by Physics). Note that there are some special purpose devices that do exist (like GPUs). They should have an efficient design meaning that they should be able to tackle as many different problems that are in same class i.e. all problems which can be expressed using the same mathematical language. Thus, there is a need for optimization. By optimization we mean using minimum physical resources in terms of time (to run as quickly as possible), space (to keep processor size/memory small), energy (to use less power) etc. required reducing some physical quantity. In terms of software: it is regarding the number of variables and instructions and memory used for running an algorithm. In terms of hardware, it is the physical implementation of how variables are mapped to physical components (component size, energy consumed, components neighbour).

Whenever we use ready made look-up tables called oracles we consider the number of times oracle is called for calculating the complexity.

There exists a hierarchy of operations connecting software and hardware. In a computer data is fed to pre-processors that converts the data to some mathematical representation (high level language) which then goes to compiler (converts instructions into low level assembly code). There is much smaller vocabulary at this level. Then, it gets converted to assembly code and then to machine code which then is seen as implementation and programmer level. All these are low level (Hardware) fixed whereas the data and pre-processing, compiler are high level (software) level and hence flexible. A living organism also shows a similar hierarchy where environment signal are picked up by sense and then brain interprets and takes actions. Here, in this case brain uses experience, takes initiative, figures out new ways. which are then send as electro-chemical signals to the nervous systems then to cells where cells take actions in the form of proteins, enzymes which is all encoded in our DNA. In computers at low level, it requires low level mapping where physical components (physical devices) must carry out both information and interpretation.

We desire to minimise resources which in turn leads to increase in noise and fluctuations/errors as limited by the Second Law of Thermodynamics and thus increase in errors. Thus, there exists a tradeoff in between the resources and errors. Hence clarity is essential. Thus, the approach for solving this problem is fixing the level of allowed errors and then to minimise resources as much as possible.

## 1.4 The coin Experiment: Analogy for intuition

Consider an unbiased coin, a preliminary high school knowledge will tell you that if you toss a coin the probability of getting heads or tails either is  $1/2$ . Given a coin we

can either toss it or look at the coin. Note that we don't know the side of the coin facing up unless and until we look at the coin. This action of looking at the coin let us call it as measurement. Thus, given a coin we can perform two operations, either measurement or toss a coin i.e. we can either see the side facing up or toss the coin. Let us now move on to the mathematical framework. Consider that we don't know the state of the coin initially, so it can be either heads or tails with equal probability of  $1/2$  and hence can be written as

$$\begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$$

where the entry in the first row corresponds to the probability of seeing heads ( $= 1/2$ ) and the entry in the second row corresponds to seeing tails ( $= 1/2$ ) upon measurement. Note that the sum of the entries in the column must be equal to 1. Now we can either look at the coin i.e. do a measurement or toss a coin. Let us toss the coin. Now since it's an unbiased coin the state of the vector will not change even after tossing the coin. Mathematically, this can be represented by a linear operation i.e. matrix-vector multiplication operation where the matrix corresponds to the operation of tossing the coin. This is shown as follows:

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

where the matrix corresponds to the operation of tossing a coin and the vector is the state vector of the coin before tossing the coin. The state vector indicates that the probability of measuring heads is  $1/2$  and probability measuring tails is  $1/2$  upon measurement. Now if we perform a measurement i.e. look at the coin the probabilistic state of the coin it will either collapse to the following

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ or } \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

for heads and tails respectively. Let us call this as basis states. Thus, if we perform another measurement the next time we will get Heads with certainty or tails with certainty depending upon the results of the previous measurement as indicated by the state vector which is also consistent with the real world results. Thus, we have a working mathematical framework of tossing a coin. Suppose, upon measurement we had found that the result of the state was head and hence the state vector was  $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ . Now when we toss the coin, we get

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

which is as we expected that now upon measurement the probability of getting heads is equal to the probability of getting tails ( $= 1/2$ ). All the mathematical framework is consistent with the real world experiences. Now, no matter how many times we toss the coin the state of the vector will remain the same i.e. probability of getting heads is equal to the probability of getting tails  $= 1/2$  unless and until we perform a measurement by looking at the coin whereupon it collapses to one of the basis states. Note that the state vector is also the eigen vector corresponding to the eigen value of 1 of the matrix (operation matrix of tossing the coin).

Let us make it more interesting by taking a biased coin. Let the probability of getting heads be  $1/4$  and of getting tails be  $3/4$ . Thus the state vector corresponding to it will be

$$\begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix}$$

We can now perform a measurement or toss the coin. Consider we don't know the state initially and we toss the coin, then the matrix-vector linear operation corresponding to the operation of tossing the coin and the state vector will be

$$\begin{pmatrix} 1/4 & 1/4 \\ 3/4 & 3/4 \end{pmatrix} \begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix} = \begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix}$$

Thus indicates that upon measurement we will get heads with a probability of  $1/4$  and tails with a probability of  $3/4$ . No matter how many times we toss the coin the state of the vector will remain the same. Upon measurement the state will collapse to one of the basis states

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ or } \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Say upon measurement we found that it was Tails i.e. the state vector collapsed to  $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ . Then again upon tossing the coin the state vector again be in a probabilistic state as follows:

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

as expected. The state of the coin after tossing will again be a probabilistic state will the probability of getting heads being  $1/4$  and the probability of tails being  $3/4$  upon measurement.

Now, let us take this a step further. Note that we can write  $\begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix} = 1/4 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 3/4 \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ . Note that we can write the state vector of the coin at any intermediate

state as a linear combination of the basis state (since the basis states are two linearly independent orthogonal vectors, they span the entire 2D Vector space and hence any vector in the 2D space can be written as a linear combination of these two vectors). Thus, we can write the state vector of coin at any time as the linear combination of the basis states as shown below:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

for  $\alpha, \beta \in \mathbb{R}$ . Thus, the state vector of the coin always exists as a linear combination of the basis states. The operation of tossing the coin is simply a matrix-vector multiplication with the state as follows.

$$\begin{aligned} \begin{pmatrix} 1/4 & 1/4 \\ 3/4 & 3/4 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} &= \alpha \begin{pmatrix} 1/4 & 1/4 \\ 3/4 & 3/4 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 1/4 & 1/4 \\ 3/4 & 3/4 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ &= \alpha \begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix} + \beta \begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix} \\ &= (\alpha + \beta) \begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix} \end{aligned}$$

Note that if we perform a measurement just after another measurement then the state of the vector remains unchanged i.e. say upon measurement we saw that the state vector was Heads then if we perform another measurement operation just after the first one then the state vector will not change and will remain as Heads. No matter the amount of time you perform consecutive measurements the state vector once collapsed to either of the basis state in the first measurement remains the same throughout. But if we perform a matrix operation (coin toss) it will then again be in a probabilistic (or linear combination sometimes called a superposition) state.

In conclusion, we saw that the state vector is a linear combination of the basis vector (with real coefficients) and there are two categories of operations we can perform on the coin either measurement or toss a coin. Similar is the case with qubits. This linear combination of basis states is what sometimes referred to as the superposition principle.

For a qubit, the state vector is a linear combination (also called superposition) of the basis states (or vectors) (with complex coefficients) and there are two types of operations we can perform on qubits either measurement (in which case it collapses to one of the basis states) or a matrix multiplication (here the matrix is unitary as a result of Axiom of Quantum mechanics we will later see). Thus, we can define the

basis vectors/states for the purpose of quantum computing as follows:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

and thus the state vector of a qubit is given as

$$|v\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

where  $\alpha, \beta \in \mathbb{C}$ . Thus, we can perform either of the two operations measurement or a matrix operation (manipulation of qubits). Upon measurement it will either collapse to either one of the basis states. The condition for manipulation of a matrix by a qubit is that it must be unitary. Thus, matrix multiplication will be as follows:

$$Uv = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} a\alpha + b\beta \\ c\alpha + d\beta \end{pmatrix}$$

where  $U$  is a unitary matrix ( $U^\dagger U = UU^\dagger = I$ ). There is another restriction that the norm of the state vector must be unit norm ( $\| |v\rangle \| = 1$ ). All these restrictions will be further explained in depth as part of axioms of Quantum Mechanics.

## 1.5 Quantum Information

We will look at Quantum information, a concept on which quantum computation is based.

### 1.5.1 A probabilistic model

Let us start classically, with a model that will probably seem completely simple to every one. Imagine that we have some physical device, called  $X$ , that has some finite, non-empty set  $\Sigma$  of possible state (classical states). For example, we might have  $\Sigma = \{0, 1\}$ , in which case we would think of  $X$  as representing a bit. For the following discussion let us restrict ourselves to this example (but keep in mind that everything can be generalized to sets other than  $\{0, 1\}$ ).

Suppose that we do not necessarily have complete information about the state of  $X$ , but instead represent our knowledge of its state by assigning probabilities to different states. For example, we might have

$$Pr[\text{state of } X \text{ is } 0] = 1/4$$

$$Pr[\text{state of } X \text{ is } 1] = 3/4$$

Mathematically we can represent this type of knowledge about the state  $X$  with a probability vector, which is a column vector whose entries are all non-negative real numbers that sum to 1. In the case at hand, the associated probability vector is

$$v = \begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix}$$

The understanding is that the entries of  $v$  are indexed by  $\Sigma$ , and when we write such a vector in the above form we are using the most natural way of ordering the elements of  $\Sigma$ .  $1/4$  entry indexed by 0,  $3/4$  entry indexed by 1.

We may write  $v[0]$  and  $v[1]$  to refer to the entries of  $v$  when necessary.

What happens when you look at  $X$ ? Of course you will not see a probability vector  $v$ . instead you will see some element of  $\Sigma$ . If our representation of the state of  $X$  by a probability vector  $v$  is in some way meaningful, you may as well imagine that the state you saw was determined randomly according to the probabilities associated with various states. Notice that by looking at the state of  $X$  you effectively change the description of your knowledge of its state. Continuing with the above example, if you look and see that the state is 0, the description of your knowledge changes from  $v$  to a new probability vector  $w$ :

$$v = \begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix} \rightarrow w = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

You know that the state is 0, and the vector  $w$  represents this knowledge. If you saw the state was 1 instead of 0, the vector would become

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

instead.

What sorts of operation can you imagine performing on  $X$ ? There are not very many deterministic operations: you could initialize  $X$  to either 0 or 1, you could not perform a NOT operation to  $X$ , or you could do nothing to  $X$  (which can still be considered an operation even though it has no effect). You could also perform an operation involving randomness - for instance perform a NOT operation with probability  $1/100$ , and otherwise do nothing. I claim that any physically meaningful operation can be represented by a matrix, with the effect of the operation being determined by the matrix-vector multiplication. For instance, these four matrices

$$INIT_0 = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}, INIT_1 = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}, NOT = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \text{and } I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

represent the deterministic operators mentioned above. For example, if our knowledge of the state of X is represented by

$$v = \begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix}$$

and we perform a NOT operation on X, the new probability vector that results is

$$w = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix} = \begin{pmatrix} 3/4 & 1/4 \end{pmatrix}$$

The probabilistic operation mentioned above is represented by the matrix

$$\begin{pmatrix} \frac{99}{100} & \frac{1}{100} \\ \frac{1}{100} & \frac{99}{100} \end{pmatrix}$$

All of these matrices have the property that (i) all entries are non-negative real numbers, and (ii) the entries of each column sum to 1. In other words, every column is a probability vector. Such matrices have a name: they are called stochastic matrices. in the simple model we are discussing, physically meaningful operations are described by stochastic matrices. it works the other way as well; any stochastic matrix describes some physically meaningful operation.

As mentioned before, this entire pictures is easily generalized to the case where  $\Sigma$  is not necessarily  $\{0, 1\}$ . In general the dimension of the vectors and matrices will be equal to the size of  $\Sigma$ .

### 1.5.2 Quantum bits (qubits)

The framework of quantum information works in a similar way to the simple probabilistic model we just saw, but with some key differences. Let us again imagine that we have a physical device called X. As before we imagine there is some set of  $\Sigma$  of possible states of X, and we will again consider for now just the simple case  $\Sigma = \{0, 1\}$ . At this point, to avoid confusion let us not refer to elements of  $\Sigma$  as *classical states*. Intuitively one can think of a classical state that you as a human can look at, touch and recognize without ambiguity. The device X will represent the quantum analogue of a bit, which we call a qubit.

We will represent out knowledge of X with column with column vectors indexed by  $\Sigma$ , but this time they will not be probability vectors. Instead of representing probability distributions, the vectors represent what we call a superposition or just

a state (by which we mean a quantum state). For example, here are a few vectors representing superpositions:

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} \frac{3}{5} \\ \frac{4}{5} \end{pmatrix}$$

Notice that the entries in these vectors are not probabilities: they are not necessarily non-negative (in fact they are not even necessarily real numbers), and they do not necessarily sum to 1. We call these numbers amplitudes instead of probabilities. The condition that replaces probabilities summing to 1 in a probability vector is this: vectors representing superpositions have Euclidian length equal to 1. In the simple case at hand where  $\Sigma = \{0, 1\}$ , this means that any vector representing a superposition has the form

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

for  $\alpha, \beta \in \mathbb{C}$  satisfying  $|\alpha|^2 + |\beta|^2 = 1$ .

Similar to the probabilistic case, if you look at the qubit X you will not see a superposition. Instead you will see either 0 or 1 just like before. The probability associated with the two possible outcomes is given by the absolute value squared of the associated amplitude-so i the superposition of X is represented by the vector

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

any you look at X, you will see 0 with probability  $|\alpha|^2$  and 1 with probability  $|\beta|^2$ . This is why we have the condition  $|\alpha|^2 + |\beta|^2$ , because the probabilities have to sum to 1 for the model to make sense. The same rules apply as for the probabilities case for determining the superposition of X after you look at it: the superposition becomes

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ or } \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

depending on whether you see 0 or 1, respectively.

So far the model does not seem qualitatively different from the probabilistic model, but that changes a lot when the possible operations that can be performed are considered. Again the possible operations are represented by matrices; but now instead of being stochastic matrices, the matrices that represent valid physical operations correspond to unitary matrices. A matrix is unitary if and only if it preserves

the Euclidian norm. Fortunately there is a very simple condition to check this: a matrix  $U$  is unitary if and only if

$$U^\dagger U = UU^\dagger = I$$

where  $U^\dagger$  is the conjugate transpose of  $U$  (meaning that you take the transpose of  $U$  and then take the complex conjugate of each of the entries). For example, these are unitary matrices:

$$H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}, I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, NOT = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, R_\theta = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

(for any real number  $\theta$  in the case of  $R_\theta$ ). For example, if  $X$  is a superposition described by

$$v = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

and the operation corresponding to the matrix  $H$  (called the hadamard transform) is performed, the superposition becomes

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

If you measured  $X$  at this point you would see outcome 0 or 1 with probability 1/2. If you didn't measure and instead applied the Hadamard transform again, the superposition would become

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

To recapitulate, these are the two things you can do to a qubit:

1. **Perform a measurement.** If the superposition of the qubit is

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

and a measurement is performed, the outcomes is 0 or 1, with probabilities  $|\alpha|^2$  and  $|\beta|^2$ , respectively. The superposition of the qubit becomes

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

depending on whether the measurement outcome was 0 or 1.

- 2. Perform a Unitary operation** For any unitary matrix  $U$ , the operation described by  $U$  transforms any superposition  $v$  into the superposition  $Uv$ .

**Example 1.5.1.** Suppose your friend has a qubit that he knows is in one of the two superpositions

$$v_0 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \text{ or } v_1 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

but he isn't sure which. How can you help him determine which one it is? Measuring right away will not help - you would see a random bit 0 or 1 with equal probability in either case. instead, you should perform a Hadamard transform and then measure. Performing the hadamard transform changes the superposition as follows:

$$Hv_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad Hv_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Now if you measure, you will see 0 (with certainty, meaning probability 1) if the original superposition was  $v_0$  and you will see 1 (with certainty) if the original superposition was  $v_1$ .

Thus instead of being just 0 and 1, quantum bits can be in a superposition between 0 and 1. Since "quantum bit" is somewhat long, researchers simply use the term "qubit" to refer to a quantum bit. Thinking of bit as vectors, a qubit can be described by a vector  $|v\rangle \in \mathbb{C}^2$ . the vector space  $\mathbb{C}^2$ . The vector space  $\mathbb{C}^2$  is also known as the state space of the qubit. An example of qubit state is

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

It turns out that in order to be a valid qubit,  $|v\rangle$  must be normalized, just as the vectors  $|0\rangle$  and  $|1\rangle$  corresponding to classical bits were indeed normalized. For the moment, let us just take this as a rule leading to the following definition.

**Definition 1.5.1. Qubit** A (pure) state of a qubit can be represented as a 2-dimensional ket vector  $|\psi\rangle \in \mathbb{C}^2$ .

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad \text{where } \alpha, \beta \in \mathbb{C} \text{ and } |\alpha|^2 + |\beta|^2 = 1$$

The condition on  $\alpha$  and  $\beta$  means that  $|\psi\rangle$  is normalized. These complex numbers  $\alpha$  and  $\beta$  are also called amplitudes of  $|\psi\rangle$ .

Throughout these lectures we will be mostly focusing on encoding information in qubits. However in general, quantum information can also be encoded in higher dimensional quantum systems. Therefore, one can similarly define a qudit as below:

**Definition 1.5.2.** A qudit, or a d-dimensional quantum system can be represented as a d-dimensional ket vector  $|\psi\rangle \in \mathbb{C}^d$

$$|\psi\rangle = \sum_{i=0}^{d-1} \alpha_i |i\rangle, \quad \text{where } \forall i, \alpha_i \in \mathbb{C} \text{ and } \sum_{i=0}^{d-1} |\alpha_i|^2 = 1$$

The condition on the coefficients  $\alpha_i$  means that  $|\psi\rangle$  is a vector of length of 1.

**Example 1.5.2.** An example of a qubit is given by the vector  $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ . The length of  $|-\rangle$  is

$$\langle -|-\rangle = \sqrt{\frac{1}{2} (1 \quad -1) \begin{pmatrix} 1 \\ -1 \end{pmatrix}} = \sqrt{\frac{1}{2}^2} = 1$$

So  $|-\rangle$  is normalized.

**Example 1.5.3.** Verify that for all values of  $\theta$ ,  $|\psi\rangle = \cos(\theta)|0\rangle + \sin(\theta)|1\rangle$  is a valid qubit state.

$$|\cos(\theta)|^2 + |\sin(\theta)|^2 = \cos^2(\theta) + \sin^2(\theta) = 1$$

Thus, clearly it is normalized for all values of  $\theta$ .

In our definition of qubits, we started from a way to write classical bits as vectors  $|0\rangle$  and  $|1\rangle$ . Note that these two vectors are orthonormal, which in the quantum notation can be expressed as  $\langle 1|0\rangle = 0$  and  $\langle 1|1\rangle = \langle 0|0\rangle = 1$ . These two vectors thus form a basis for  $\mathbb{C}^2$ , in that any vector  $|v\rangle \in \mathbb{C}^2$  can be written as  $|v\rangle = \alpha|0\rangle + \beta|1\rangle$  for some coefficients  $\alpha, \beta \in \mathbb{C}$ . This basis corresponding to "classical" bits is used so often that it carries a special name:

1.

**Definition 1.5.3. Computational Basis/Standard Basis:** Consider the 2-dimensional complex vector space  $\mathbb{C}^2$ . The standard basis, or sometimes known as the computational basis,  $\mathcal{S} = \{|0\rangle, |1\rangle\}$  is an orthonormal basis for this vector space, where the basis vectors are

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

In general, for a d-dimensional complex vector space  $\mathbb{C}^d$ . The standard basis or computational basis states are represented as  $|0\rangle, |1\rangle, \dots, |d-1\rangle$  where

$$|i\rangle = \begin{pmatrix} \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix}$$

where it is a column vector of size  $2^d \times 1$  with 1 at the ith location and 0 elsewhere.

Of course, there might be many other bases for  $\mathbb{C}^2$ . But here, we will discuss only those basis which are most frequently used.

2.

**Definition 1.5.4. Hadamard Basis:** The Hadamard basis is an orthonormal basis in which the states are represented as  $\mathcal{H} = |+\rangle, |-\rangle$  where

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{and} \quad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

3. **Y Basis:** The basis in which the states are represented as  $|+i\rangle, |-i\rangle$  where

$$|+i\rangle = \frac{|0\rangle + i|1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ i \end{pmatrix} \quad \text{and} \quad |-i\rangle = \frac{|0\rangle - i|1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -i \end{pmatrix}$$

**Example 1.5.4.** We now verify that the hadamard basis is indeed orthonormal using the bra-ket notation as shown below:

$$\langle +|+\rangle = \frac{1}{2} (1 \ 1) \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{2} 2 = 1 \implies \sqrt{\langle +|+ \rangle} = 1$$

So  $|+\rangle$  is normalized. Furthermore, the inner product

$$\langle +|-\rangle = \frac{1}{2} (1 \ 1) \begin{pmatrix} 1 \\ -1 \end{pmatrix} = 0$$

So  $|+\rangle$  and  $|-\rangle$  are orthogonal to each other.

**Example 1.5.5.** Express  $|1\rangle$  in the Hadamard basis. That is, find coefficients  $\alpha$  and  $\beta$  such that  $|1\rangle = \alpha|+\rangle + \beta|-\rangle$ . Clearly, putting  $\alpha = \frac{1}{\sqrt{2}}$  and  $\beta = \frac{1}{\sqrt{2}}$  satisfies the system of linear equations.

The linear superposition  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  is part of the private world of electron. For us to know the electron's state, we must make a measurement. making a measurement gives us a single classical bit of information 0 or 1. The simplest measurement is in the standard basis, and measuring  $|\psi\rangle$  in this  $\{|0\rangle, |1\rangle\}$  basis yields 0 with probability  $|\alpha|^2$ , and 1 with probability  $|\beta|^2$ .

One important aspect of the measurement process is that it alters the state of the qubit: the effect of the measurement is that the new state is exactly the outcome of the measurement. i.e., if the outcome of the measurement of  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  yields 0, then following the measurement, the qubit is in the state  $|0\rangle$ . This implies that you cannot collect any additional information about  $\alpha, \beta$  by repeating the measurement.

More generally, we may choose any orthogonal basis  $\{|v\rangle, |w\rangle\}$  and measure the qubit in that basis. To do this, we rewrite our state in that basis:  $|\psi\rangle = \alpha'|v\rangle + \beta'|w\rangle$ . The outcome is  $v$  with probability  $|\alpha'|^2$ , and  $w$  with probability  $|\beta'|^2$ . If the outcome of the measurement on  $|\psi\rangle$  yields  $|v\rangle$ , then as before, the qubit is then in state  $|v\rangle$ .

### 1.5.3 Example of Qubits

#### Atomic Orbitals

The electrons within an atom exist in quantized energy levels. Qualitatively these electronic orbits (or "orbitals" as we like to call them) can be thought of as resonating standing waves, in close analogy to the vibrating waves one observes on a tightly held piece of string. Two such levels can be isolated to configure the basis states for a qubit.

#### Photon Polarization

Classically, a photon may be described as a travelling electromagnetic wave. This description can be fleshed out using Maxwell's equation, but for our purposes we will focus simply on the fact that an electromagnetic wave has a polarization which describes the orientation of the electric field oscillations (see figure). So, for a given direction of photon motion, the photon's polarization axis might lie anywhere in a 2-d plane perpendicular to that motion. It is thus natural to pick an orthonormal 2-d basis (such as  $\vec{x}$  and  $\vec{y}$ , or "vertical" and "horizontal") to describe the polarization state (i.e. polarization direction) of a photon. In a quantum mechanical description,

this 2-d nature of the photon polarization is represented by a qubit, where the amplitude of the overall polarization state in each basis vector is just the projection of the polarization in that direction.

The polarization of a photon can be measured by using a polaroid film or a calcite crystal. A suitably oriented sheet transmits x-polarized photons and absorbs y-polarized photons. Thus a photon that is in a superposition  $|\phi\rangle = \alpha|x\rangle + \beta|y\rangle$  is transmitted with probability  $|\alpha|^2$ . If the photon now encounters another polaroid sheet with the same orientation, then it is transmitted with probability 1. On the other hand, if the second polaroid sheet has its axes crossed at right angles to the first one, then if the photon is transmitted by the first polaroid, then it is definitely absorbed by the second sheet. This pair of polarized sheets at right angles thus blocks all the light. A somewhat counter-intuitive result is not obtained by interposing a third polaroid sheet at a 45 degree angle between the first two. Now a photon that is transmitted by the first sheet makes it through the next two with some non zero probability.

To see this first observe that any photon transmitted through the first filter is in the state,  $|0\rangle$ . The probability this photon is transmitted through the second filter is  $1/2$  since it is exactly the probability that a qubit in the state  $|0\rangle$  ends up in the state  $|+\rangle$  when measured in the  $|+\rangle, |-\rangle$  basis. We can repeat this reasoning for the third filer, except now we have a qubit in state  $|+\rangle$  being measured in the  $|0\rangle, |1\rangle$  basis - the change that the outcome is  $|0\rangle$  is once again  $1/2$ .

## Spins

Like photon polarization, the spin of a (spin-1/2) particle is a two-state system, and can be described by a qubit. Very roughly speaking, the spin is a quantum description of the magnetic moemnt of an electron which behaves like a spinning charge. The two allowed states can roughly be thought of as clockwise rotations ("spin-up") and counter clockwise rotations ("spin-down").

### 1.5.4 Multiple Qubits

In order to talk about what happens when we have multiple qubits, it will be helpful to briefly return to the probabilistic model from before. Suppose that X and Y are devices that implement bits. Then there are 4 possible states of the pair (X,Y), namely 00, 01, 10 and 11. Thus, our set of states  $\Sigma$  corresponding to this pair is now  $\{00, 01, 10, 11\}$ . In the probabilistic model we represent our knowledge of the state of the pair (X,Y) with a 4 dimensional probability vector. For example we could have

the following probability vector

$$\begin{pmatrix} \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{2} \\ 0 \\ \frac{3}{8} \end{pmatrix}$$

with the first entry indicating the probability associated with state 00, second entry denoting the probability associated with state 01, third entry denoting the probability associated with state 10, fourth entry denoting the probability associated with state 11. (The vector indices are labelled by the states in the order given by binary notation.) Operations again correspond to stochastic matrices, but this time the matrices are  $4 \times 4$  matrices. For example, the matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

is stochastic. It happens to correspond to the operation where you do nothing if the first bit is 0 but if the first bit is 1 then replace the second bit with a random bit.

The quantum variant works in an analogous way. If we have two qubits  $(X, Y)$ , then a superposition of these two qubits is a 4-dimensional vector with Euclidian length equal to 1. For example:

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{i}{2} \\ -\frac{1}{2} \end{pmatrix}$$

Measurement works the same way as before, except that the outcome will be two bits. For example, measuring the previous superposition gives results as follows:

$$00 \text{ with probability } \left| \frac{1}{\sqrt{2}} \right|^2 = \frac{1}{2}$$

$$01 \text{ with probability } |0|^2 = 0$$

$$10 \text{ with probability } \left| \frac{i}{\sqrt{2}} \right|^2 = \frac{1}{4}$$

$$11 \text{ with probability } \left| -\frac{1}{2} \right|^2 = \frac{1}{4}$$

We can also measure only one qubit out of the two bits leaving the other alone, which is called partial measurement. We will later on see what happens in case of

partial measurement. Unitary operations also work the same way as before, but this time are  $4 \times 4$  matrices. For example, there is a 2 qubit unitary operation called controlled-NOT:

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

The same pattern is used for 3 qubits, 4 qubits etc. The dimension of the vectors and matrices grows exponentially: 8 dimensional vectors for 3 qubits, 16 dimensional vectors for 4 qubits, etc.

By the way there is no reason you cannot consider the model for any other choice of  $\Sigma$ , instead of  $\Sigma$  corresponding to all possible strings of a given length. Typically, however, we will focus on the case where  $\Sigma = \{0, 1\}^n$  for positive integer  $n$ .

Classically, if we have two bits, we write them as '00', '01', '10', '11'. But how can we write two qubits? One strategy is to again associate each of them two classical bits  $x_1, x_2 \in \{0, 1\}^2$  with a vector. labelling the first qubit A and the second one B, we could perform the mapping from strings to orthonormal vectors as

$$0_A 0_B \rightarrow |00\rangle_{AB} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad 0_A 1_B \rightarrow |01\rangle_{AB} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$1_A 0_B \rightarrow |10\rangle_{AB} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad 1_A 1_B \rightarrow |11\rangle_{AB} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Note that the resulting vectors are in  $\mathbb{C}^4$  with dimension  $d = 2^2 = 4$ , where the dimension corresponds to the number of possible strings. It turns out that one can write a two-qubit state  $|\psi\rangle_{AB} \in \mathbb{C}^4$  as a superposition of these vectors, where we again demand that  $|\psi\rangle_{AB}$  is normalized. As an example, let us consider a state

$|\psi\rangle_{AB}$  that is an equal superposition of all above standard basis vectors:

$$\begin{aligned} |\psi\rangle_{AB} &= \frac{1}{2} |00\rangle_{AB} + \frac{1}{2} |01\rangle_{AB} + \frac{1}{2} |10\rangle_{AB} + \frac{1}{2} |11\rangle_{AB} \\ &= \frac{1}{2} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \frac{1}{2} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \frac{1}{2} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \frac{1}{2} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \\ &= \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \end{aligned}$$

The sum of amplitudes  $\frac{1}{2}$  squared is  $4 \frac{1}{2^2} = 1$ , therefore  $|\psi\rangle$  is a valid two qubit quantum state. As you might have guessed, we now proceed analogously when considering  $n$  qubits. To address multiple qubits, we first look at the vector representation for multiple classical bits. For binary strings of length  $n$ , consider the vector space  $\mathbb{C}^n$ , where each coordinate is labelled by a string  $x = x_1, \dots, x_n$ . There are a total of  $d = 2^n$  such strings, so we can label each string  $x$  with a different integer  $i \in [1, d]$ . We can then express the string  $x$  as a vector  $|x_i\rangle$  that is 0 everywhere, except at the position labelled by  $i$ . A quantum state of  $n$  qubits can then be written as

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$$

with  $\alpha_x \in \mathbb{C}$  and  $\sum_x |\alpha_x|^2 = 1$ . The numbers  $\alpha_x$  are again called amplitudes. We emphasize that the dimension of the vector space  $\mathbb{C}^{2^n}$  increases exponentially with the number  $n$  of bits. The space  $\mathbb{C}^d$  with  $d = 2^n$  is thereby called the state space of  $n$  qubits. This means that we need an exponential number of parameters  $\alpha_x$  to keep track of only  $n$  qubits, in sharp contrast to the  $n$  parameters  $x_1, \dots, x_n$  to describe  $n$  classical bits. You might wonder whether this was the only way to write down qubits. After all, we had simply chosen some mapping from strings of length  $n$  to vectors in  $\mathbb{C}^d$ . Could we have chosen any other mapping from strings to vectors? It turns out that the answer to this is yes - as long as each string gets mapped to a vector that is orthonormal to the others. The mapping above, however, is very convenient and generally adopted within the realm of quantum computing. Analogous to the case of a single qubit, the basis given by the set of vectors  $|x\rangle |x \in \{0, 1\}^n$  is called the *standard/computational basis*.

**Definition 1.5.5. Standard basis for  $n$  qubits.** Consider the state space of  $n$  qubits  $\mathbb{C}^d$ , where  $d = 2^n$ . For each distinct string  $x \in \{0, 1\}^n$ , associate  $x$  with a

distinct integer  $i \in \{1, 2, \dots, d\}$ . The standard basis for  $\mathbb{C}^d$  is an orthonormal basis given by  $\mathcal{S}_n = \{|x\rangle |x \in \{0, 1\}^n\}$  where  $|x\rangle$  are d-dimensional vectors.

$$|x\rangle = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

where 1 is at the  $i$ th position.

Let us summarize our discussion in the following definition of an  $n$  qubit quantum state.

**Definition 1.5.6.** An  $n$ -qubit state  $|\psi\rangle \in \mathbb{C}^d$  with  $d = 2^n$  can be written as a superposition of standard basis elements.

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle, \quad \text{where } \forall x, \alpha_x \in \mathbb{C} \text{ and } \sum_{x \in \{0,1\}^n} |\alpha_x|^2 = 1$$

Let us now consider two examples of two qubit states. The first is so famous it carries a special name and we will see it very frequently in the course of these notes.

**Example 1.5.6.** Example 0.2.2 Consider two qubits A and B, in the two qubit state known as the EPR (Einstein, Podolsky and Rosen) pair, one can label the joint state as AB

$$|EPR\rangle_{AB} = \frac{1}{\sqrt{2}}(|00\rangle_{AB} + |11\rangle_{AB}) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \frac{1}{2} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

which is an equal superposition between the vectors  $|00\rangle_{AB}$  and  $|11\rangle$ . The length of this vector is given by the (square root o) inner product

$$\begin{aligned} \langle EPR | EPR \rangle_{AB} &= \frac{1}{\sqrt{2}}(\langle 00 |_{AB} + \langle 11 |_{AB}) \cdot \frac{1}{\sqrt{2}}(|00\rangle_{AB} + |11\rangle_{AB}) \\ &= \frac{1}{2}(\langle 00 | 00 \rangle_{AB} + \langle 00 | 11 \rangle_{AB} + \langle 11 | 00 \rangle_{AB} + \langle 11 | 11 \rangle_{AB}) \\ &= \frac{1}{2} \cdot 2 = 1 \implies \sqrt{\langle EPR | EPR \rangle} = 1 \end{aligned}$$

**Example 1.5.7.** Consider the two qubit state

$$|\psi\rangle_{AB} = \frac{1}{\sqrt{2}}(|01\rangle_{AB} + |11\rangle_{AB}) \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

For this state, the second qubit always corresponds to bit 1. We will later see that this is significantly different state compared to  $|EPR\rangle_{AB}$ . It is not entangled!!.

### Tensor Products: How to combine qubits

Returning again briefly to the probabilistic model, let us suppose that as before X and Y are devices implementing bits, and the two devices are completely uncorrelated with one another—let us say that the probability vector corresponding to X is

$$v = \begin{pmatrix} \frac{2}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{pmatrix}$$

and the probability vector corresponding to Y is

$$w = \begin{pmatrix} \frac{1}{4} \\ \frac{3}{4} \\ \frac{2}{4} \\ \frac{1}{4} \end{pmatrix}$$

Then the 4 dimensional probability vector corresponding to the pair (X, Y) is easily determined by multiplying the corresponding probabilities. In particular, the resulting vector is

$$v \otimes w = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{12} \\ \frac{2}{12} \\ \frac{1}{12} \\ \frac{1}{4} \end{pmatrix}$$

The operation  $\otimes$  is called the Kronecker product or the tensor product. (It is most common in quantum computing to use the term tensor product to refer to this operation.) In general, for any two matrices

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1l} \\ b_{21} & b_{22} & \dots & b_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k1} & b_{k2} & \dots & b_{kl} \end{pmatrix}$$

we define  $A \otimes B$  to be the  $nk \times ml$  matrix

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1m}B \\ a_{21}B & a_{22}B & \dots & a_{2m}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}B & a_{n2}B & \dots & a_{nm}B \end{pmatrix}$$

The definition works for vectors by thinking of them as matrices with only one column. The tensor product satisfies many nice properties. For example, it is an associative operation;

$$(A \otimes B) \otimes C = A \otimes (B \otimes C)$$

for any choice of matrices A, B and C. Thus, it makes sense to talk about products such as  $A \otimes B \otimes C \otimes \dots \otimes Z$  without including parentheses, because it doesn't matter in which order the products are evaluated. Next, we have

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$$

for any choice of matrices A, B, C and D (assuming the sizes of the matrices are such that the products AC and BD make sense). The distributive law holds for tensor products;

$$A \otimes (B + C) = A \otimes B + A \otimes C \quad \text{and} \quad (A + B) \otimes C = A \otimes C + B \otimes C$$

Also, for matrices A and B and any scalar  $\alpha$ , we have

$$(\alpha A) \otimes B = A \otimes (\alpha B) = \alpha(A \otimes B)$$

In other words, scalars “float freely” through the tensor product. A word of warning, however, is that the tensor product is not commutative; in general it may be the case that  $A \otimes B \neq B \otimes A$ .

Not every probability vector  $v$  representing a distribution of  $(X, Y)$  can be written as a tensor product. For example,

$$v = \begin{pmatrix} 1/2 \\ 0 \\ 0 \\ 1/2 \end{pmatrix}$$

cannot be written as a tensor product. In this distribution we have

$$\Pr[\text{state of } (X, Y) \text{ is } 00] = \Pr[\text{state of } (X, Y) \text{ is } 11] = \frac{1}{2}$$

We say that X and Y are correlated in this case. The only way a probability vector can be written as a tensor product is when the associated systems are uncorrelated (or independent). As you might have guessed, we do exactly the same thing in the quantum case as in the classical, probabilistic case. If X and Y are qubits having associated superpositions

$$v = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad \text{and} \quad w = \begin{pmatrix} \gamma \\ \delta \end{pmatrix}$$

then the superposition of the pair (X, Y) is

$$v \otimes w = \begin{pmatrix} \alpha\gamma \\ \alpha\delta \\ \beta\gamma \\ \beta\delta \end{pmatrix}$$

The superposition

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

is an example of a superposition that cannot be written as a tensor product. In the quantum case, this type of correlation between X and Y is special and we call it entanglement. We will talk about entanglement a lot during the course.

For example,  $|1010\rangle$  is a 16 dimensional vector with a 1 in the position indexed by 1010 in binary (which is the eleventh entry because we start with 0000). The vector

$$\frac{1}{\sqrt{2}}|000000\rangle + \frac{1}{\sqrt{2}}|111111\rangle$$

would be written

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \vdots \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

in the usual vector notation. An arbitrary vector with entries indexed by  $\{0, 1\}^n$ , which perhaps refers to a superposition of n qubits, can again be written as a linear combination of the elements in the basis

$$\{|x\rangle : x \in \{0, 1\}^n\}$$

for instance as

$$|\phi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$$

**Example 1.5.8.** Let us suppose that we have two qubits X and Y in the superposition

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

Using the Dirac notation we write this superposition as

$$\frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle$$

For every ket  $|\psi\rangle$  there is a corresponding object  $\langle\psi|$ , called a “bra”. You may think that this is a strange name for a mathematical object, but the names “bra” and “ket” are derived from the fact that when you put a bra and a ket together, you get a “bracket”. For this to make sense you need to know what a bra is—for any vector  $|\psi\rangle$  we define

$$\langle\psi| = (\langle\psi|)^\dagger$$

which is the conjugate transpose of  $|\psi\rangle$ . In other words,  $\langle\psi|$  is the row vector you get by transposing  $|\psi\rangle$  and taking the conjugate of each of its entries. For instance:

$$|\psi\rangle = \begin{pmatrix} \frac{1+\iota}{2} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \implies \langle\psi| = \begin{pmatrix} \frac{1-\iota}{2} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

Now, when you juxtapose a bra and a ket, the implicit operation is matrix multiplication (thinking of the vectors as matrices with only one row or one column). A row vector times a column vector results in a scalar, and this scalar will be the inner product (or bracket) of the vectors involved. For instance, if

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad \text{and} \quad |\phi\rangle = \begin{pmatrix} \gamma \\ \delta \end{pmatrix}$$

then

$$\langle\psi|\phi\rangle = \langle\psi| |\phi\rangle = (\bar{\alpha} \quad \bar{\beta}) \begin{pmatrix} \gamma \\ \delta \end{pmatrix} = \bar{\alpha}\gamma + \bar{\beta}\delta$$

When you have an expression such as

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$$

It is easy to express  $\langle \psi |$  using similar notation; it is

$$\langle \psi | = \sum_{x \in \{0,1\}^n} \overline{\alpha_x} |x\rangle$$

When you juxtapose a ket and a bra in the opposite order, such as

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

you do not get a scalar—a column vector times a row vector gives you a matrix. It is easy to determine the action of this matrix on another vector. For instance,

$$|\psi\rangle \langle \phi| |\gamma\rangle = |\psi\rangle \langle \phi| \gamma = \langle \phi | \gamma \rangle |\psi\rangle$$

Later on when we wish to speak at a higher level of abstraction about computational problems, algorithms, etc., we may refer to  $-xi$  where  $x$  is some arbitrary mathematical object (such as a matrix, a graph, or a list of numbers). In this case the interpretation is that we are implicitly referring to the encoding of  $x$  with respect to some agreed upon encoding scheme.

Let's imagine that we have two qubits, A and B. We know that we can describe the state of A as  $|\psi\rangle_A$  and the one of B as  $|\phi\rangle_B$ . How can we write down the combined state  $|\psi\rangle_{AB}$  of A and B together? The rule for computing the joint state is given by the so-called tensor product (sometimes also called Kronecker product). For two qubits

$$|\psi\rangle_A = \alpha_A |0\rangle_A + \beta_A |1\rangle_A = \begin{pmatrix} \alpha_A \\ \beta_A \end{pmatrix}$$

$$|\phi\rangle_B = \alpha_B |0\rangle_B + \beta_B |1\rangle_B = \begin{pmatrix} \alpha_B \\ \beta_B \end{pmatrix}$$

the joint state  $|\psi\rangle_{AB} \in \mathbb{C}^2 \otimes \mathbb{C}^2$  can be expressed as the tensor product of individual vectors  $|\psi\rangle_A$  and  $|\phi\rangle_B$

$$|\psi\rangle_{AB} = |\psi\rangle_A \otimes |\phi\rangle_B = \begin{pmatrix} \alpha_A \\ \beta_A \end{pmatrix} \otimes \begin{pmatrix} \alpha_B \\ \beta_B \end{pmatrix} = \begin{pmatrix} \alpha_A \alpha_B \\ \alpha_A \beta_B \\ \beta_A \alpha_B \\ \beta_A \beta_B \end{pmatrix}$$

As you may have guessed, we can of course also combine the state of two quantum systems A and B if they are larger than just one qubit. The general definition of the tensor product of two vectors is given by

**Definition 1.5.7.** Given two vectors  $|\psi_1\rangle \in \mathbb{C}^{d_1}$  and  $|\psi_2\rangle \in \mathbb{C}^{d_2}$  respectively, the tensor product is given by

$$|\psi_1\rangle \otimes |\psi_2\rangle = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_d \end{pmatrix} \otimes |\psi_2\rangle = \begin{pmatrix} \alpha_1 |\psi_2\rangle \\ \vdots \\ \alpha_d |\psi_2\rangle \end{pmatrix}$$

and  $|\psi_1\rangle \otimes |\psi_2\rangle$  lies in the state space  $\mathbb{C}^{\mathbb{K}} \otimes \mathbb{C}^{d_2}$ .

The following is simplified (or rather, lazy) notation are commonly used in Quantum Information: Omitting the tensor product symbol:  $|\psi\rangle_A \otimes |\psi\rangle_B = |\psi\rangle_A |\psi\rangle_B$ . Writing classical bit as a string:  $|0\rangle_A \otimes |0\rangle_B = |0\rangle_A |0\rangle_B = |0\rangle_{AB}$ . Combing several identical states:  $|\psi\rangle_1 \otimes |\psi\rangle_2 \dots \otimes |\psi\rangle_n = |\psi\rangle^{\otimes n}$ .

**Proposition 1.5.1.** *The tensor product satisfies several useful properties:*

1. **Distributive:**  $|\psi_1\rangle \otimes (|\psi_2\rangle + |\psi_3\rangle) = |\psi_1\rangle \otimes |\psi_2\rangle + |\psi_1\rangle \otimes |\psi_3\rangle$ . Similarly,  $(|\psi_1\rangle + |\psi_2\rangle) \otimes |\psi_3\rangle = |\psi_1\rangle \otimes |\psi_3\rangle + |\psi_2\rangle \otimes |\psi_3\rangle$ .
2. **Associative:**  $|\psi_1\rangle \otimes (|\psi_2\rangle \otimes |\psi_3\rangle) = (|\psi_1\rangle \otimes |\psi_2\rangle) \otimes |\psi_3\rangle$ .
3. **NOT Commutative:** In general,  $|\psi_1\rangle \otimes |\psi_2\rangle \neq |\psi_2\rangle \otimes |\psi_1\rangle$  unless of course  $|\psi_1\rangle = |\psi_2\rangle$ .

These relations not only hold for kets, but also for bras.

To understand the definition of the tensor product, let us have a look at a few examples. The first relates to the definition of the standard basis for multiple qubits. Indeed, you may have been wondering, if we could have proceeded in a somewhat less ad hoc manner than starting from classical strings  $x \in \{0, 1\}^n$  and assigning to them vectors  $|x\rangle$  in a space of dimension  $d = 2^n$ . Indeed, you may have started to wonder why  $n$  qubits resulted in a state space of a dimension that is exponential in  $n$  in the first place. The reason for this, is that the law of quantum mechanics tells us that the state space of two quantum systems is indeed combined by the tensor product.

**Example 1.5.9.** Let's recover the standard basis of two qubits, from the standard basis of the individual qubits using the tensor product rule. Recall that the standard basis for two qubits AB is given by

$$|00\rangle_{AB} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, |01\rangle_{AB} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, |10\rangle_{AB} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, |11\rangle_{AB} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

This basis can be constructed, by taking the tensor product of standard basis elements for individual qubits:  $|0\rangle_A \otimes |0\rangle_B, |0\rangle_A \otimes |1\rangle_B, |1\rangle_A \otimes |0\rangle_B, |1\rangle_A \otimes |1\rangle_B$ . For example, consider

$$|1\rangle_A \otimes |0\rangle_B = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes |0\rangle_B = \begin{pmatrix} 0 & |0\rangle_B \\ 1 & |0\rangle_B \end{pmatrix} = |10\rangle_{AB}$$

We have already seen a few other examples of two qubit states. Let's see whether we can recover them from two individual qubit states using the tensor product.

**Example 1.5.10.** Consider the states  $|+\rangle_A = \frac{1}{\sqrt{2}}(|0\rangle_A + |1\rangle_A)$  and  $|1\rangle_B$ . The joint state  $|\psi\rangle_{AB}$  is given by

$$|\psi\rangle_{AB} = |+\rangle_A \otimes |1\rangle_B = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \otimes |1\rangle_B = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & |1\rangle_B \\ 1 & |1\rangle_B \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

One can also express the joint state in the standard basis by:

$$\begin{aligned} |\psi\rangle_{AB} &= \frac{1}{\sqrt{2}}(|0\rangle_A + |1\rangle_A) \otimes |1\rangle_B \\ &= \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |1\rangle_B + |1\rangle_A \otimes |1\rangle_B) \\ &= \frac{1}{\sqrt{2}}(|01\rangle_{AB} + |11\rangle_{AB}) \end{aligned}$$

**Example 1.5.11.** Consider the states  $|+\rangle_A = \frac{1}{\sqrt{2}}(|0\rangle_A + |1\rangle_A)$  and  $|+\rangle_B = \frac{1}{\sqrt{2}}(|0\rangle_B + |1\rangle_B)$ . The joint state  $|\psi\rangle_{AB}$  is

$$\begin{aligned} |\psi\rangle_{AB} &= \frac{1}{\sqrt{2}}(|0\rangle_A + |1\rangle_A) \otimes \frac{1}{\sqrt{2}}(|0\rangle_B + |1\rangle_B) \\ &= \frac{1}{2}(|00\rangle_{AB} + |01\rangle_{AB} + |10\rangle_{AB} + |11\rangle_{AB}) \\ &= \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \end{aligned}$$

This is the state which is in an equal superposition of all the standard basis vectors for the two qubits.

The following is an example of a state that can actually not be expressed as the tensor product of two qubit states. Such states are rather special, and play an important role later in our course. Nevertheless, let's have a look at it to see how we might also express a two qubit state in different bases.

**Example 1.5.12.** Consider the state

$$|\psi\rangle_{AB} = \frac{1}{\sqrt{2}}(|+\rangle_A|+\rangle_B + |-\rangle_A|-\rangle_B)$$

Let us express this state in terms of the standard basis, by expanding the terms

$$|\psi\rangle_A |\psi\rangle_B = \frac{1}{2}(|0\rangle_A + |1\rangle_A)(|0\rangle_B + |1\rangle_B) = \frac{1}{2}(|00\rangle_{AB} + |10\rangle_{AB} + |01\rangle_{AB} + |11\rangle_{AB})$$

$$|-\rangle_A |-\rangle_B = \frac{1}{2}(|0\rangle_A - |1\rangle_A)(|0\rangle_B - |1\rangle_B) = \frac{1}{2}(|00\rangle_{AB} - |10\rangle_{AB} - |01\rangle_{AB} + |11\rangle_{AB})$$

Substituting this into equation gives

$$\begin{aligned} |\psi\rangle_{AB} &= \frac{1}{\sqrt{2}}(ket+_A|+\rangle_B + |-\rangle_A|-\rangle_B) \\ &= \frac{1}{2\sqrt{2}}(|00\rangle_{AB} + |10\rangle_{AB} + |01\rangle_{AB} + |11\rangle_{AB} + |00\rangle_{AB} + |00\rangle_{AB} - |01\rangle_{AB} - |10\rangle_{AB} + |11\rangle_{AB}) \\ &= \frac{1}{\sqrt{2}}(|00\rangle_{AB} + |11\rangle_{AB}) = |EPR\rangle_{AB} \end{aligned}$$

where  $|EPR\rangle_{AB}$  is the state we have seen previously. We see that the coefficients of  $|EPS\rangle_{AB}$  are the same whether we write it in the Hadamard basis or the standard basis.

## 1.6 Measurements

Let us consider what happens if we measure a qubit. Classically, you can think of the measurement of a bit as simply a readout: we have a system that encodes the state '0' and '1' and we make a measurement to find out which one it is.

### 1.6.1 Simple Measurements

This linear superposition  $|\psi\rangle = \sum_j \alpha_j |j\rangle$  is part of the private world of the electron. Access to the information describing this state is severely limited —

in particular, we cannot actually measure the complex amplitudes  $\alpha_j$ . This is not just a practical limitation; it is enshrined in the measurement postulate of quantum physics.

A measurement on this  $k$  state system yields one of at most  $k$  possible outcomes: i.e. an integer between 0 and  $k - 1$ . Measuring  $|\psi\rangle$  in the standard basis yields  $j$  with probability  $|\alpha_j|^2$ .

One important aspect of the measurement process is that it alters the state of the quantum system: the effect of the measurement is that the new state is exactly the outcome of the measurement. I.e., if the outcome of the measurement is  $j$ , then following the measurement, the qubit is in state  $|j\rangle$ . This implies that you cannot collect any additional information about the amplitudes  $\alpha_j$  by repeating the measurement.

Intuitively, a measurement provides the only way of reaching into the Hilbert space to probe the quantum state vector. In general this is done by selecting an orthonormal basis  $|e_0\rangle, \dots, |e_{k-1}\rangle$ . The outcome of the measurement is  $|e_j\rangle$  with probability equal to the square of the length of the projection of the state vector  $\psi$  on  $|e_j\rangle$ . A consequence of performing the measurement is that the new state vector is  $|e_j\rangle$ . Thus measurement may be regarded as a probabilistic rule for projecting the state vector onto one of the vectors of the orthonormal measurement basis.

### Measurement in Standard Basis

Let's first consider a single qubit. Quantum measurements can result in probabilistic outcomes, highlighting that quantum information and classical information really are fundamentally different. For example, if the state  $|\psi\rangle \in \mathbb{C}^2$  is a superposition between  $|0\rangle$  and  $|1\rangle$ , then upon measuring  $|\psi\rangle$ , we obtain different measurement outcomes corresponding to some probability distribution. How are such probabilities generated? The probability of different outcomes, for instance for outcome ‘0’, can be computed by, roughly speaking, “looking at how much ‘0’ is actually in our qubit vector”. This is quantified by the inner product between  $|\psi\rangle$  and  $|0\rangle$ . More concretely, consider a single qubit state  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ , where  $\alpha, \beta$  are complex numbers. Upon measuring the qubit, one obtains the outcome “0” with probability  $p_0$  and “1” with probability  $p_1$ . These probabilities can be determined by computing the inner products

$$p_0 = |\langle\psi|0\rangle|^2 = \left| (\alpha^* \quad \beta^*) \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right|^2 = |\alpha|^2$$

$$p_1 = |\langle\psi|1\rangle|^2 = \left| (\alpha^* \quad \beta^*) \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right|^2 = |\beta|^2$$

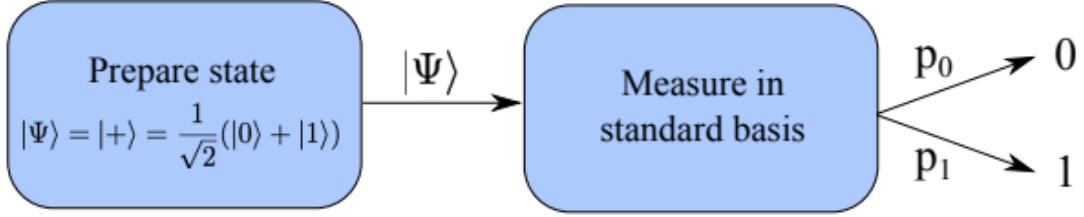


Figure 1.3: Genuine randomness from the preparation of a qubit in superposition

We now see a good reason for the condition  $|\alpha|^2 + |\beta|^2 = 1$ : it means that  $p_0 + p_1 = 1$ , that is, the probabilities of observing ‘0’ and ‘1’ add up to one. In quantum computer science, it is customary to label the outcomes ‘0’ for “ $|0\rangle$ ” and ‘1’ for “ $|1\rangle$ ” and more generally  $x$  for outcomes  $|x\rangle$ , while in physics people often use +1 for “ $|0\rangle$ ” and -1 for “ $|1\rangle$ ”.

### Application: Randomness from a deterministic process

Can we do anything interesting with what we have learned so far? It turns out the answer is yes: by preparing just single qubits and measuring in the standard basis, we can in principle achieve a task that it is impossible classically. Namely, we can produce true random numbers. Consider the following process illustrated in Figure 1: first, prepare a qubit in the state  $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ . Next, measure this state in the standard basis. The probability of obtaining each outcome can then be calculated by evaluating the inner products:

$$p_0 = |\langle +|0\rangle|^2 = \left| \frac{1}{\sqrt{2}}(\langle 0| + \langle 1|) |0\rangle \right|^2 = \left| \frac{1}{\sqrt{2}}(\langle 0|0\rangle + \langle 1|0\rangle) \right|^2 = \frac{1}{\sqrt{2}}^2 = \frac{1}{2}$$

$$p_1 = |\langle +|1\rangle|^2 = \left| \frac{1}{\sqrt{2}}(\langle 0| + \langle 1|) |1\rangle \right|^2 = \left| \frac{1}{\sqrt{2}}(\langle 0|1\rangle + \langle 1|1\rangle) \right|^2 = \frac{1}{\sqrt{2}}^2 = \frac{1}{2}$$

This simple example already tells us something about the power of quantum information: We could build a machine that deterministically prepares the qubit  $|+\rangle$ , followed by a measurement in the standard basis. Since  $p_0 = p_1 = 1/2$ , this machine allows us to produce a perfect random number - even though no randomness has been used inside our machine! In contrast, one can prove that no classical deterministic machine can produce random numbers from scratch.

We saw how to measure a single qubit in the standard basis. The rule for computing probabilities of measurement outcomes generalizes in a direct way to measuring

n-qubit states. Indeed, consider an n-qubit quantum state

$$|\Psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$$

What happens when  $|\Psi\rangle$  is measured in the standard basis  $|x\rangle_x$ ? It turns out that the probability of outcome  $x$  is given by  $p_x = |\langle x|\Psi\rangle|^2 = |\alpha_x|^2$ , explaining again the need for normalization of the vector  $|\Psi\rangle$ .

### Measuring a qubit in other bases

Can we measure our qubit in any other basis? The answer to this is yes! Indeed this is another feature that distinguishes quantum from classical, where the only basis around is the standard basis. To find out how to analyze such a more general setting, let us first take a step back and consider how we found the probabilities above. When measuring in the standard basis, the probabilities are given by the squared amplitudes when writing out the state in terms of the standard basis. When measuring a qubit in a different orthonormal basis, given by vectors  $\mathcal{G} = \{|v\rangle, |v^\perp\rangle\}$ , it is intuitive that we would have to express the qubit in the new basis. That is, we need to find amplitudes  $\hat{\alpha}$  and  $\hat{\beta}$  such that

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \hat{\alpha}|v\rangle + \hat{\beta}|v^\perp\rangle$$

**Example 1.6.1.** As an example, let consider again the qubit  $|+\rangle = (1/\sqrt{2})(|0\rangle + |1\rangle)$ . Instead of measuring it in the standard basis, let us now measure in the basis  $H = \{|+\rangle, |-\rangle\}$  given by the two orthonormal vectors  $|+\rangle$  and  $|-\rangle = (1/\sqrt{2})(|0\rangle - |1\rangle)$ . Clearly, we can write the qubit as  $1|+\rangle + 0|-\rangle$ . Thus the probability of obtaining measurement outcome “ $|+\rangle$ ” is 1. We thus see that the probabilities of measurement outcomes depends dramatically on the basis in which we measure.

**Example 1.6.2.** Consider measuring an arbitrary qubit  $\alpha|0\rangle + \beta|1\rangle$  in the basis  $\{|+\rangle, |-\rangle\}$ . To find out how to express the qubit in this other basis, it is convenient to determine how the basis elements  $|0\rangle$  and  $|1\rangle$  look like in this basis. We find that

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

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

We thus have

$$\begin{aligned}\alpha|0\rangle + \beta|1\rangle &= \frac{1}{\sqrt{2}}[\alpha(|+\rangle + |-\rangle) + \beta(|+\rangle - |-\rangle)] \\ &= \frac{\alpha + \beta}{\sqrt{2}}|+\rangle + \frac{\alpha - \beta}{\sqrt{2}}|-\rangle\end{aligned}$$

This means that we obtain outcome "|+" with probability  $|\alpha + \beta|^2/2$  and outcome "|-" with probability  $|\alpha - \beta|^2/2$ .

**Example 1.6.3.** Consider the state  $|\Psi\rangle = |0\rangle$ . What are the probabilities  $p_0, p_1$  for measuring  $|\Psi\rangle$  in the standard basis? What are the probability  $p_+, p_-$  for measuring  $|\Psi\rangle$  in the Hadamard basis? For measuring in the standard basis the probabilities  $p_0 = 1, p_1 = 0$ . The probabilities in the Hadamard basis are  $p_+ = \frac{1}{2}, p_- = \frac{1}{2}$ .

Quite often we do not care about the entire probability distribution, but just the probability of one specific outcome. Is there a more efficient way to find this probability than to rewrite the entire state  $|\psi\rangle$  in another basis? To investigate this, let us consider a single qubit

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

Remember that the elements of the standard basis are orthonormal. This means that

$$(|0\rangle)^\dagger|0\rangle = (1 \ 0) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1,$$

$$(|0\rangle)^\dagger|1\rangle = (1 \ 0) \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 0$$

Because the vectors are orthonormal, we could thus have found the desired probabilities by simply computing the inner product between two vectors, as claimed above. Specifically, when given the qubit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  we obtain outcomes "|0⟩" and "|1⟩" with probabilities

$$p_0 = |\langle 0|\psi\rangle|^2 = \left| (1 \ 0) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \right|^2 = |\alpha|^2$$

$$p_1 = |\langle 1|\psi\rangle|^2 = \left| (0 \ 1) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \right|^2 = |\beta|^2$$

**Example 1.6.4.** Suppose we measure  $|0\rangle$  in the hadamard basis  $\mathcal{H}$  (see above). The probabilities of observing outcomes " $|+\rangle$ " and " $|-\rangle$ " are given by

$$p_+ = |\langle 0|\psi \rangle|^2 = \left| \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \right|^2 = \frac{1}{2}$$

$$p_- = |\langle 1|\psi \rangle|^2 = \left| \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \right|^2 = \frac{1}{2}$$

For multiple qubits, the rules for finding probabilities is analogous.

**Definition 1.6.1.** Suppose that we measure a quantum state  $|\psi\rangle$  in the orthonormal basis  $\{|b_j\rangle\}_{j=1}^d$ . The probability of observing outcome " $b_j$ " can be found by computing

$$p_j = |\langle b_j|\psi \rangle|^2$$

The post-measurement state when obtaining outcome " $b_j$ " is given by  $|b_j\rangle$ .

Let us now consider some examples to gain intuition on measuring quantum systems in different bases. First, let us have a look at a single qubit example.

**Example 1.6.5.** Consider the qubit  $|\Psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + \iota|1\rangle)$ , and measure the qubit in the  $\{|+\rangle, |-\rangle\}$  basis. the probabilities of obtaining "+" and "-" can be evaluated as follows:

$$\begin{aligned} p_+ &= |\langle \Psi | + \rangle|^2 = \left| \frac{1}{2} (\langle 0| - \iota \langle 1|) (|0\rangle + |1\rangle) \right|^2 \\ &= \frac{1}{4} \left| \langle 0|0\rangle + \langle 0|1\rangle - \iota \langle 1|0\rangle - \iota \langle 1|1\rangle \right|^2 \\ &= \frac{1}{4} |1 - \iota|^2 \\ &= \frac{1}{4} (1 - \iota)(1 + \iota) = \frac{1}{2} \end{aligned}$$

$$\begin{aligned} p_- &= |\langle \Psi | - \rangle|^2 = \left| \frac{1}{2} (\langle 0| - \iota \langle 1|) (|0\rangle - |1\rangle) \right|^2 \\ &= \frac{1}{4} \left| \langle 0|0\rangle + \langle 0|1\rangle - \iota \langle 1|0\rangle + \iota \langle 1|1\rangle \right|^2 \\ &= \frac{1}{4} |1 + \iota|^2 \\ &= \frac{1}{4} (1 - \iota)(1 + \iota) = \frac{1}{2} \end{aligned}$$

This example shows that when the states involved have complex-valued amplitudes, one has to take extra caution when evaluating the inner product: namely when taking the bra  $\langle \Psi |$ , one should remember to alter the +/- sign whenever a complex number is involved (since the bra  $\langle \Psi |$  is the conjugate transpose of the ket  $|\Psi\rangle$ ).

While we will generally talk about n-qubits, we can of course also consider a quantum system comprised of three levels  $|0\rangle$ ,  $|1\rangle$ , and  $|2\rangle$ , i.e. a qutrit. The rule for obtaining the probabilities of measurement outcomes remains unchanged.

**Example 1.6.6.** Consider a qutrit, which is a 3-dimensional quantum system represented by the vector

$$|v\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

and measure in the basis  $\mathcal{B} = \{|b_1\rangle, |b_2\rangle, |b_3\rangle\}$  where

$$|b_1\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad |b_2\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \quad |b_3\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$$

The probabilities of obtaining each outcome can be calculated as follows:

$$\begin{aligned} p_{b_1} &= |\langle b_1 | v \rangle|^2 = \frac{1}{2} \\ p_{b_2} &= |\langle b_2 | v \rangle|^2 = \langle b_2 | v \rangle \langle v | b_2 \rangle = \frac{1}{2\sqrt{2}}(1+1)\frac{1}{2\sqrt{2}}(1+1) = \frac{1}{2} \\ p_{b_3} &= |\langle b_3 | v \rangle|^2 = \langle b_3 | v \rangle \langle v | b_3 \rangle = \frac{1}{2\sqrt{2}}(1-1)\frac{1}{2\sqrt{2}}(1-1) = 0 \end{aligned}$$

### Measurement Example I: Phase Estimation

Now that we have discussed qubits in some detail, we can are prepared to look more closely at the measurement principle. Consider the quantum state, —

$$|\psi\rangle = \frac{1}{\sqrt{2}} |0\rangle + \frac{e^{i\theta}}{\sqrt{2}} |1\rangle$$

If we were to measure this qubit in the standard basis, the outcome would be 0 with probability 1/2 and 1 with probability 1/2. This measurement tells us only about

the norms of the state amplitudes. Is there any measurement that yields information about the phase,  $\theta$ ? To see if we can gather any phase information, let us consider a measurement in a basis other than the standard basis, namely

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

What does  $|\phi\rangle$  look like in this new basis? This can be expressed by first writing,

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

Now we are equipped to rewrite  $|\psi\rangle$  in the  $\{|+\rangle, |-\rangle\}$ -basis,

$$\begin{aligned} |\psi\rangle &= \frac{1}{\sqrt{2}}|0\rangle + \frac{e^{i\theta}}{\sqrt{2}}|1\rangle \\ &= \frac{1}{2}(|+\rangle + |-\rangle) + \frac{e^{i\theta}}{\sqrt{2}}(|+\rangle - |-\rangle) \\ &= \frac{1+e^{i\theta}}{2}|+\rangle + \frac{1-e^{i\theta}}{2}|-\rangle \end{aligned}$$

Recalling the Euler relation,  $e^{i\theta} = \cos\theta + i\sin\theta$ , we see that the probability of measuring  $|+\rangle$  is  $\frac{1}{4}((1 + \cos\theta)^2 + \sin^2\theta) = \cos^2(\theta/2)$ . A similar calculation reveals that the probability of measuring  $|-\rangle$  is  $\sin^2(\theta/2)$ . Measuring in the  $(|+\rangle, |-\rangle)$ -basis therefore reveals some information about the phase  $\theta$ . Later we shall show how to analyze the measurement of a qubit in a general basis.

### Measurement example II: general Qubit Bases

What is the result of measuring a general qubit state  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ , in a general orthonormal basis  $|v\rangle, |v^\perp\rangle$ , where  $|v\rangle = a|0\rangle + b|1\rangle$  and  $|v^\perp\rangle = b^*|0\rangle - a^*|1\rangle$ ? You should also check that  $|v\rangle$  and  $|v^\perp\rangle$  are orthogonal by showing that  $\langle v|v^\perp\rangle = 0$ .

To answer this question, let us make use of our recently acquired braket notation. We first show that the states  $|v\rangle$  and  $|v^\perp\rangle$  are orthogonal, that is, that their inner product is zero:

$$\begin{aligned} \langle v|v^\perp\rangle &= (b^*|0\rangle - a^*|1\rangle)^\dagger(a|0\rangle + b|1\rangle) \\ &= (b\langle 0| - a\langle 1|)^\dagger(a|0\rangle + b|1\rangle) \\ &= ba\langle 0|0\rangle - a^2\langle 1|0\rangle + b^2\langle 0|1\rangle - ab\langle 1|1\rangle \\ &= ba - 0 + 0 - ab \\ &= 0 \end{aligned}$$

Here we have used the fact that  $\langle i|j\rangle = \delta_{ij}$ .

Now, the probability of measuring the state  $|\psi\rangle$  and getting  $|v\rangle$  as a result is,

$$\begin{aligned} p_\psi(v) &= |\langle v|\psi\rangle|^2 \\ &= |(a^* \langle 0| + b^* \langle 1|)(\alpha|0\rangle + \beta|1\rangle)|^2 \\ &= |a^*\alpha + b^*\beta|^2 \end{aligned}$$

Similarly,

$$\begin{aligned} P_\psi(v^\perp) &= |\langle v^\perp|\psi\rangle|^2 \\ &= |(b\langle 0| - a\langle 1|)(\alpha|0\rangle + \beta|1\rangle)|^2 \\ &= |b\alpha - a\beta|^2 \end{aligned}$$

### Expectation Values

Physicists (but also computer scientists!) like to compute expectation values of measurement outcomes, as they provide an indication of the average behavior, if one was to perform a measurement many times (however we shall see later, that the measurement will perturb the state!). Let us suppose that we measure a qubit  $|\Psi\rangle$  in the standard basis  $\{|0\rangle, |1\rangle\}$ . We will adopt the physics convention of labelling outcomes  $\pm 1$ . This means that we associate the outcome " $|0\rangle$ " with outcome  $+1$ , and outcome " $|1\rangle$ " with outcome  $-1$ . The expectation value the outcome obtained when measuring  $|\psi\rangle$  is then

$$E = 1|\langle 0|\psi\rangle|^2 - 1\langle 1|\psi\rangle|^2$$

Note that since  $|\langle 0|\psi\rangle|^2 = \langle\psi|0\rangle\langle 0|\psi\rangle$ , we have

$$E = \langle\psi|(|0\rangle\langle 0| - |1\rangle\langle 1|)|\psi\rangle = \langle\psi|Z|\psi\rangle$$

where  $Z = |0\rangle\langle 0| - |1\rangle\langle 1|$ . As we shall see later,  $Z$  is called the Pauli-Z matrix.

### 1.6.2 Measuring Multiple Systems

We saw how to measure some quantum state  $|\psi\rangle$ . Let us now consider what happens if we measure the state of multiple qubits, where we think of measuring each qubit in a separate basis. To understand this, it is useful to realize that a basis for the joint state space  $\mathbb{C}_A^{d_A} \otimes \mathbb{C}_B^{d_B}$  can be obtained from bases for the individual state spaces  $\mathbb{C}_A^{d_A}$  and  $\mathbb{C}_B^{d_B}$ . Specifically, if  $\{|b_j^A\rangle\}_j$  is a basis for  $\mathbb{C}_A^{d_A}$  and  $\{|b_j^B\rangle\}_j$  is a basis for the state space  $\mathbb{C}_B^{d_B}$ , then the set of vectors  $\{|b_j^A\rangle \otimes |b_k^B\rangle\}_{j=1}^{d_A} \}_{k=1}^{d_B}$  gives a basis for  $\mathbb{C}_A^{d_A} \otimes \mathbb{C}_B^{d_B}$ .

**Example 1.6.7.** Consider the basis  $\{|0\rangle_A, |1\rangle_A\}$  for qubit A, and the basis  $\{|+\rangle_B, |-\rangle_B\}$  for qubit B. A basis for the joint state AB is then given by  $\{|0\rangle_A|+\rangle_B, |0\rangle_A|-\rangle_B, |1\rangle_A|+\rangle_B, |1\rangle_A|-\rangle_B\}$ .

Let us not think how we might construct some measurement for two quantum states from measurements of the individual ones. Suppose we measure particle A in the basis  $\{|b_j^A\rangle\}_j$  and particle B in the basis  $\{b_k^B\}_k$  when the joint state of both particles is given by  $|\psi\rangle_{AB}$ . What is the probability that we obtain outcome " $|b_j^A\rangle$ " on A, and outcome " $|b_k^B\rangle$ " on B? To find such joint probabilities, we first write down the joint basis for quantum states A and B as above:  $\{\{|b_j^A\rangle|b_k^B\rangle\}_j\}_k$ . We can apply the usual rule to compute the probability as

$$p_{jk} = |\langle b_j^A | \langle b_k^B | \psi \rangle_{AB} \rangle|^2$$

**Example 1.6.8.** Consider the two qubits in an EPR pair

$$|EPR\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

and measure them both in the standard basis. The probabilities of obtaining outcomes 00, 01, 10 and 11 are given by

$$p_{00} = p_{11} = \frac{1}{2}$$

$$p_{01} = p_{10} = 0$$

### Partial Measurements

Suppose we have a system consisting of two or more qubits and we only measure one of them. For example, suppose the qubits are X and Y, and these qubits are in the state

$$|\psi\rangle = \frac{1}{2}|00\rangle - \frac{i}{2}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

We know what the distribution of measurement outcomes would be if we measured both qubits:

$$Pr[\text{outcome is } 00] = \frac{1}{4} \quad Pr[\text{outcome is } 01] = 0$$

$$Pr[\text{outcome is } 10] = \frac{1}{4} \quad pr[\text{outcome is } 11] = \frac{1}{2}$$

The probability that a measurement of the first qubit, for instance, results in outcome 0 should therefore be  $1/4 + 0 = 1/4$  and the probability of outcome 1 should be

$1/4 + 1/2 = 3/4$ . Intuitively, this should be the case regardless of whether or not the second qubit was actually measured. Indeed this is the case.

However, what is different between the case where the second qubit is measured and the case where it is not is the superposition of the system after the measurement (or measurements). If both qubits are measured, the superposition will be one of  $|00\rangle$ ,  $|01\rangle$ ,  $|10\rangle$ , or  $|11\rangle$ , depending on the measurement outcome. If only the first qubit is measured, however, this may not be the case. I'll explain how you determine the state of the system after the measurement of the first qubit for the example above, and the method for a general case should be clear. We begin by considering the possible outcome 0 for the measurement. We consider the state

$$|\psi\rangle = \frac{1}{2}|00\rangle - \frac{\iota}{2}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

and cross off all of the terms in the sum that are inconsistent with measuring a 0 for the first qubit:

$$|\psi\rangle = \frac{1}{2}|00\rangle$$

which leaves

$$\frac{1}{|00\rangle}$$

The probability of measuring 0 is the norm-squared of this vector.

$$Pr[\text{measurement outcomes is 0}] = |\frac{1}{2}|00\rangle|^2 = \frac{1}{4}$$

and the state of the two qubits after the measurement conditioned on the measurement outcome being 0 is the vector itself renormalized:

$$\frac{1}{2}|00\rangle \|\frac{1}{2}|00\rangle\| = |00\rangle$$

The possible measurement outcome 1 is handled similarly. we cross off the terms is inconsistent with measuring a 1 which leaves

$$-\frac{\iota}{2}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

The probability of measuring a 1 is then the norm-squared of this vector,

$$Pr[\text{measurement outcome is 1}] = \left\| -\frac{\iota}{2}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle \right\|^2 = \left| -\frac{\iota}{2} \right|^2 + \left| \frac{1}{\sqrt{2}} \right|^2 = \frac{3}{4}$$

and conditioned on measuring a 1 is then the norm-squared of this vector

$$\frac{-\frac{\iota}{2}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle}{\left\| -\frac{\iota}{2}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle \right\|} = -\frac{\iota}{\sqrt{3}}|10\rangle + \sqrt{\frac{2}{3}}|11\rangle$$

A completely equivalent way of performing this calculation is to begin by writing

$$|\psi\rangle = |0\rangle|\phi_0\rangle + |1\rangle|\phi_1\rangle$$

for some choice of  $|\phi_0\rangle$  and  $|\phi_1\rangle$ . (There will always be a unique choice for each that works.) in this case we have

$$|\psi\rangle = |0\rangle\left(\frac{1}{2}|0\rangle\right) + |1\rangle\left(-\frac{\iota}{2}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right)$$

so  $|\phi_0\rangle = \frac{1}{2}|0\rangle$  and  $|\phi_1\rangle = -\frac{\iota}{2}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ . The probabilities for the two measurement outcomes are as follows:

$$Pr[\text{outcome is } 0] = \|\phi_0\|^2 = \left|\frac{1}{2}\right|^2 = \frac{1}{4}$$

$$Pr[\text{outcome is } 1] = \|\phi_1\|^2 = \left|-\frac{\iota}{2}\right|^2 + \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{3}{4}$$

Conditioned on the measurement outcome, the state of the two qubits is as follows:

$$\text{measurement outcome is } 0 \implies \text{state becomes } \frac{1}{\|\phi_0\|}|0\rangle|\phi_0\rangle = |00\rangle$$

$$\text{measurement outcome is } 1 \implies \text{state becomes } \frac{1}{\|\phi_1\|}|1\rangle|\phi_1\rangle = \frac{-\iota}{\sqrt{3}}|10\rangle + \sqrt{\frac{2}{3}}|11\rangle$$

The same method works for more than 2 qubits as well.

**Example 1.6.9.** Suppose 3 qubits are in the superposition

$$|\psi\rangle = \frac{1}{2}|000\rangle + \frac{1}{2}|100\rangle + \frac{1}{2}|101\rangle - \frac{1}{2}|111\rangle$$

and the third qubit is measured. What are the probabilities of the two possible measurement outcomes and what are the resulting superpositions of the three qubits for each case? To determine the answer we rewrite,

$$|\psi\rangle = \left(\frac{1}{2}|00\rangle + \frac{1}{2}|10\rangle\right)|0\rangle + \left(\frac{1}{2}|10\rangle - \frac{1}{2}|11\rangle\right)|1\rangle$$

The probability that the measurement outcome is 0 is

$$\left\| \frac{1}{2} |00\rangle + \frac{1}{2} |10\rangle \right\|^2 = \frac{1}{2}$$

and in this case the resulting superposition is

$$\sqrt{2} \left( \frac{1}{2} |00\rangle + \frac{1}{2} |10\rangle \right) |0\rangle = \frac{1}{\sqrt{2}} |000\rangle + \frac{1}{\sqrt{2}} |100\rangle$$

The probability that the measurement outcome is 1 is

$$\left\| \frac{1}{2} |10\rangle - \frac{1}{2} |11\rangle \right\|^2 = \frac{1}{2}$$

and in this case the resulting superposition is

$$\sqrt{2} \left( \frac{1}{2} |10\rangle - \frac{1}{2} |11\rangle \right) |1\rangle = \frac{1}{\sqrt{2}} |101\rangle - \frac{1}{\sqrt{2}} |111\rangle$$

# Chapter 2

## Axioms of Quantum Mechanics

### 2.1 Postulate 1: The State Space Postulate

The state of an isolated quantum mechanical system is completely specified by a state vector  $|\psi\rangle$  which is a unit vector that belongs to a complex vector space with inner product (a Hilbert Space)  $\mathbb{C}^n$ , where  $n$  is the dimension of the state space of the system.

Consider some physical system that can be in  $N$  different, mutually exclusive classical states (it means a state in which the system can be found if we observe it)  $|0\rangle, |1\rangle, \dots, |N-1\rangle$ . Then, a *pure quantum state* (or *state*)  $|\psi\rangle$  is a superposition of classical states, written as:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle + \dots + \alpha_{N-1} |N-1\rangle = \sum_{i=0}^{N-1} \alpha_i |i\rangle$$

where  $\alpha_i$  are complex numbers called amplitude of  $|i\rangle$  in  $|\psi\rangle$ . A system in quantum state  $|\psi\rangle$  is in a superposition of the classical states  $|0\rangle, |1\rangle, \dots, |N-1\rangle$  with amplitudes  $\alpha_0, \alpha_1, \dots, \alpha_{N-1}$  respectively. Mathematically, the states  $|0\rangle, |1\rangle, \dots, |N-1\rangle$  forms an  $N$ -dimensional Hilbert space (i.e.  $N$  dimensional complex vector space with inner product defined). We write the quantum state  $|\psi\rangle$  as a  $N$ -dimensional column vector of its amplitudes in the Hilbert Space.

$$|\psi\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{N-1} \end{pmatrix}$$

Consider a Hilbert space  $\mathcal{H}_A$  with orthonormal basis  $|0\rangle, |1\rangle, \dots, |N-1\rangle$  and another Hilbert space  $\mathcal{H}_B$  with an orthonormal basis  $|0'\rangle, |1'\rangle, \dots, |M-1\rangle$ . Then we can combine the two Hilbert spaces using tensor products to form a new Hilbert space  $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$  which is a  $N \times M$  dimensional Hilbert space with basis  $|i\rangle \otimes |j\rangle$  where  $|i\rangle$  is a basis vector of  $\mathcal{H}_A$  ( $|i\rangle \in \{0, \dots, N-1\}$ ) and  $|j\rangle$  is a basis vector of  $\mathcal{H}_B$  ( $|j\rangle \in \{0', \dots, M-1\}$ ). The basis vectors of  $\mathcal{H}$  are  $|i\rangle \otimes |j\rangle$  where  $i \in \{0, \dots, N-1\}$  and  $j \in \{0', \dots, M-1\}$ . Thus, an arbitrary state of  $\mathcal{H}$  can be written as:

$$|\psi\rangle = \sum_{i=0}^{N-1} \sum_{j=0'}^{M-1} \alpha_{ij} |i\rangle \otimes |j\rangle$$

where  $\alpha_{ij}$  are complex numbers called amplitudes of  $|i\rangle \otimes |j\rangle$  in  $|\psi\rangle$ . Such a state is called *bipartite*. Similarly, this can be extended to tripartite states which is a tensor product of three Hilbert spaces and so on.

The simplest system is qubit - a quantum mechanical system with two-dimensional state space. Suppose  $|0\rangle$  and  $|1\rangle$  are two orthonormal basis vectors also generally called as **computational basis** of the state space of the qubit. Then the arbitrary state of the qubit can be represented as

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

where  $\alpha$  and  $\beta$  are complex numbers such that  $|\alpha|^2 + |\beta|^2 = 1$  (since it's a unit vector). The condition  $|\alpha|^2 + |\beta|^2 = 1$  is called the *normalization condition*. The complex numbers  $\alpha$  and  $\beta$  are called probability amplitudes for the states  $|0\rangle$  and  $|1\rangle$  respectively.

Intuitively, one can think of the state  $|0\rangle$  and  $|1\rangle$  as the analogous to the classical bit states 0 and 1. The qubit differs from the classical bit in the sense that the qubit can exist in a superposition of the states  $|0\rangle$  and  $|1\rangle$ , i.e. of the form  $\alpha|0\rangle + \beta|1\rangle$  where  $\alpha$  and  $\beta$  are complex numbers, which is neither in state  $|0\rangle$  or in  $|1\rangle$  (or in both the states simultaneously) which does not happen for a classical bit. This is the essence of quantum superposition which can be taken advantage of in quantum computing.

**Example 2.1.1.** A qubit can exist in a state such as

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

which is a superposition of the states  $|0\rangle$  and  $|1\rangle$  with amplitude  $1/\sqrt{2}$  for being in state  $|0\rangle$  and amplitude  $-1/\sqrt{2}$  for being in the state  $|1\rangle$ .

**Example 2.1.2.** Say a state of a qubit is  $|\psi\rangle = 7|0\rangle + (3 + 5i)|1\rangle$ . In the matrix form, it can be written as:

$$|\psi\rangle = \begin{pmatrix} 7 \\ 3 + 5i \end{pmatrix}$$

In general we say that any linear combination  $\sum_i \alpha_i |\psi_i\rangle$  is a superposition of the states  $|\psi_i\rangle$  with amplitudes  $\alpha_i$  for the state  $|\psi_i\rangle$ .

**There are two things we can do with a quantum state: Measure it or let it evolve (without measuring it).**

## 2.2 Postulate 2: Evolution Postulate

The time evolution of a closed Quantum System is described by the Schrodinger equation as:

$$i\hbar \frac{d}{dt} |\psi(t)\rangle = H |\psi(t)\rangle$$

where  $|\psi(t)\rangle$  is the state of the system at time  $t$ ,  $H$  is the Hamiltonian operator of the system and  $\hbar$  is the reduced Planck's constant. The Hamiltonian operator is an operator that corresponds to the total energy of the system. The Schrodinger equation is a linear differential equation and is the quantum analog of Newton's second law of motion. The solution to the Schrodinger equation gives the state of the system at any time  $t$  given the initial state of the system. If we know the Hamiltonian of a system, we can predict the future state of the system. For the purpose of Quantum computing, we are interested in the time evolution of a quantum system. Thus, we will always assume that the Hamiltonian of a quantum system is known. Thus, simplifying the Schrodinger equation, we can write it as:

$$\frac{d}{dt} |\psi(t)\rangle = -\frac{i}{\hbar} H |\psi(t)\rangle$$

Rearranging the above equation, we get:

$$\int_{\psi_{t_1}}^{\psi_{t_2}} \frac{d|\psi\rangle}{|\psi\rangle} = - \int_{t_1}^{t_2} \frac{i}{\hbar} H dt$$

Integrating, the above equation we get,

$$\ln \left( \frac{|\psi_{t_2}\rangle}{|\psi_{t_1}\rangle} \right) = -\frac{i}{\hbar} H(t_2 - t_1)$$

$$|\psi_{t_2}\rangle = e^{-\frac{i}{\hbar}H(t_2-t_1)} |\psi_{t_1}\rangle$$

Now we know that the Hamiltonian operator  $H$  is Heritian and as proved in the appending we know that  $e^A$  is unitary for any Heritian operator  $A$ . Thus, the time evolution operator  $U(t_2, t_1) = e^{-\frac{i}{\hbar}H(t_2-t_1)}$  is unitary. Hence, we can write the time evolution of a quantum state as:

$$|\psi(t_2)\rangle = U(t_1, t_2) |\psi(t_1)\rangle$$

where  $U$  is a Unitary time evolution Operator. Thus, we have described a discrete time desciption of the dynamics using unitary operators. For the purpose of Quantum Computing, we will deal with discrete time evolution of the quantum system described by the exponenetial of the Hamiltonian (Hermitian operator) which is a unitary operator as shown and not the continuous time evolution of the quantum system described by the hamiltonian. Thus, we restate the postulate for Quantum Computing as: **The evolution of a closed quantum system is described by a Unitary transformation. That is, the two states at times  $t_1$  and time  $t_2$  are related by a unitary operator  $U(t_2, t_1)$  such that  $|\psi(t_2)\rangle = U(t_2, t_1) |\psi(t_1)\rangle$ .**

Note that in quantum computing we often apply a unitary operator to a closed quantum state. This is contradictory statement. The act of applying a unitary operator is in itself an external interaction with the quantum system, thus it no longer remains closed. It turns out, that it is possible to write down a time-varying Hamiltonian for a Quantum system, in which the hamiltonian for the system is not constant, but varies according to some external control parameters which are under experimentalist's control, and could be changed during the course of an experiment. The system is therefore not closed, but it does evolve according to Schrödinger's equation with a time varying Hamiltonian, to some good approximation. Thus we will describe the evolutions of a quantum system even those system's which aren't closed using unitary operators.

## 2.3 Postulate 3: Measurement Postulate

### Measurement in the computational basis

Consider a Quantum System in a state  $|\psi\rangle = \sum_{j=0}^{N-1} \alpha_j |j\rangle$  which is some unknown superposition of the classical states  $|j\rangle$ . We cannot know exactly what the superposition of the states  $|\psi\rangle$  is since we can see only the classical states. Thus if we measure the state  $|\psi\rangle$  it will collapse to one of the classical states (say  $|j\rangle$ ). To which classical state will it collapse to (which will we see)? This is not known in advance,

we can only predict the probability of the state collapsing to a particular classical state. The probability of the state  $|\psi\rangle$  collapsing to the classical state  $|j\rangle$  is given by **Born's Rule** as:

$$P(j) = |\langle j|\psi \rangle|^2 = |\alpha_j|^2$$

where  $\alpha_j$  is the amplitude of the state  $|j\rangle$  in the state  $|\psi\rangle$ . Thus, the probability of the state  $|\psi\rangle$  collapsing to the classical state  $|j\rangle$  is the square of the amplitude of the state  $|j\rangle$  in the state  $|\psi\rangle$ . Now, since it's a probability distribution this implies that the sum of the probabilities of the state  $|\psi\rangle$  collapsing to any of the classical states  $|j\rangle$  is 1. Thus, we have:

$$\sum_j P(j) = \sum_j |\alpha_j|^2 = 1$$

thus,  $\alpha_j$  are called probability amplitudes. (**Note that since operations on a quantum state are unitary in nature and a Unitary operator as shown in the appendix preserves norm, thus once the amplitudes are normalized to satisfy the Born's rule we can be sure that the probabilities will remain normalized no matter the number of times we perform any unitary operation on the quantum state.**) Once we measure the state  $|\psi\rangle$  and it collapses to the classical state  $|j\rangle$ , the state of the system is now  $|j\rangle$  and all the information that might have been contained in the amplitudes of  $\alpha_i$  is gone.

Note that the probabilities of various measurement outcomes are exactly the same when we measure  $|\psi\rangle$  or when we measure the state  $e^{i\theta}|\psi\rangle$ . The term  $e^{i\theta}$  is called the **global phase** and thus has no physical significance.

Note that when the experimentalist and their experimental apparatus observes the system, it is an external physical system in other words, and the interaction makes the quantum system no longer closed and thus not necessarily subject to the unitary evolution. Thus, more formally we can state the postulate as:

**Quantum measurements are described by a collection  $\{M_m\}$  of measurement operators.** These are operators acting on the state space  $|\psi\rangle$  being measured. The index  $m$  refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is  $|\psi\rangle$  immediately before the measurement then the probability that result  $m$  occurs is given by:

$$p(m) = \langle \psi | M_m^\dagger M_m | \psi \rangle$$

and the state of the system after the measurement is:

$$\frac{M_m |\psi\rangle}{\sqrt{p(m)}}$$

Now since the sum of the probabilities of all the measurement outcomes is 1, we have:

$$1 = \sum_m p(m) = \sum_m \langle \psi | M_m^\dagger M_m | \psi \rangle$$

Thus, the measurement operators satisfy the completeness equation:

$$\sum_m M_m^\dagger M_m = I$$

**Example 2.3.1.** Measurement of a qubit in the computational basis. Measurement of a qubit with two outcomes defined by the two measurement operators  $M_0 = |0\rangle\langle 0|$  and  $M_1 = |1\rangle\langle 1|$  (note that both the measurement operators are Hermitian and  $M_0^2 = M_0, M_1^2 = M_1$ , thus follow Completeness relation). Suppose the qubit is in the state  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ . The probability of the qubit collapsing to the state  $|0\rangle$  is:

$$p(0) = |\langle 0|\psi \rangle|^2 = |\alpha|^2$$

and the probability of the qubit collapsing to the state  $|1\rangle$  is:

$$p(1) = |\langle 1|\psi \rangle|^2 = |\beta|^2$$

Thus, the state of the qubit after the measurement is:

$$\frac{M_0 |\psi\rangle}{\sqrt{p(0)}} = \frac{\alpha |0\rangle}{|\alpha|}$$

$$\frac{M_1 |\psi\rangle}{\sqrt{p(1)}} = \frac{\beta |1\rangle}{|\beta|}$$

**Example 2.3.2.** Consider a state  $|\psi\rangle = \left(\frac{1}{\sqrt{6}} - \frac{1}{\sqrt{3}}\right)|+\rangle + \left(\frac{1}{\sqrt{6}} + \frac{1}{\sqrt{3}}\right)|-\rangle$  (it can be given in any basis, say here it is given in hadamard basis). Find the probabilities of measurement in Hadamard basis and Standard basis.

Probability of measuring the state  $|\psi\rangle$  in Hadamard basis to be  $|+\rangle$  is:

$$p(+)=|\langle +|\psi \rangle|^2 = \left| \left( \frac{1}{\sqrt{6}} - \frac{1}{\sqrt{3}} \right) \right|^2 = \frac{1}{6} + \frac{1}{3} - \frac{2}{\sqrt{18}} = \frac{3-2\sqrt{2}}{6}$$

Probability of measuring the state  $|\psi\rangle$  in Hadamard basis to be  $|-\rangle$  is:

$$p(-)=|\langle -|\psi \rangle|^2 = \left| \left( \frac{1}{\sqrt{6}} + \frac{1}{\sqrt{3}} \right) \right|^2 = \frac{1}{6} + \frac{1}{3} + \frac{2}{\sqrt{18}} = \frac{3+2\sqrt{2}}{6}$$

Probability of measuring the state  $|\psi\rangle$  in Standard basis to be  $|0\rangle$  is:

$$p(0) = |\langle 0|\psi \rangle|^2 = \left| \frac{1}{\sqrt{2}} \left( \frac{2}{\sqrt{6}} \right) \right|^2 = \left| \frac{1}{\sqrt{3}} \right|^2 = \frac{1}{3}$$

Probability of measuring the state  $|\psi\rangle$  in Standard basis to be  $|1\rangle$  is:

$$p(1) = |\langle 1|\psi \rangle|^2 = \left| \frac{1}{\sqrt{2}} \left( \frac{2}{\sqrt{3}} \right) \right|^2 = \left| \frac{\sqrt{2}}{\sqrt{3}} \right|^2 = \frac{1}{3}$$

### Important Note

#### Distinguishing Non-Orthogonal States

**Theorem 2.3.1.** *Non-orthogonal states can't be reliably distinguished.*

Consider two people Alice and Bob. Alice prepares a quantum state  $|\psi_i\rangle$  ( $1 \leq i \leq n$ ) from some fixed set of states known to both parties and sends it to Bob. Bob has to determine which state Alice sent him.

Suppose the states  $|\psi_i\rangle$  are orthonormal. Then Bob can measure the state  $|\psi_i\rangle$  using the following procedure. Define Measurement operators,  $M_i = |\psi_i\rangle \langle \psi_i|$ , one for each possible index  $i$ , and an additional measurement operator  $M_0 = I - \sum_{i=1}^n |\psi_i\rangle \langle \psi_i|$ . The measurement operators satisfy the completeness relation  $\sum_{i=0}^n M_i^\dagger M_i = I$ . Thus, for a state  $|\psi_i\rangle$  then  $p(i) = \langle \psi_i | M_i | \psi_i \rangle = 1$ , so the result  $i$  occurs with certainty. Thus, it is possible to reliably distinguish the orthonormal states  $|\psi_i\rangle$ .

Suppose the states  $|\psi_i\rangle$  are not orthonormal. then we can prove that there is no quantum measurement capable of distinguishing the quantum states. The idea is that Bob will do a measurement described by measurement operators  $M_j$  with outcome  $j$ . Depending on the outcome of the measurement Bob tries to guess what the index  $i$  was using some rule,  $i = f(j)$  (where  $f(\cdot)$  is the rule to make the guess). Bob can't distinguish between the states  $|\psi_1\rangle$  and  $|\psi_2\rangle$  because  $|\psi_2\rangle$  can be decomposed into a (non-zero) component parallel to  $|\psi_1\rangle$ , and a component orthogonal to  $|\psi_1\rangle$ .

### 2.3.1 Projective Measurements

A special case of the general measurement postulate. Consider a Quantum System with quantum state  $|\psi\rangle$ . A projective measurement on some space, with  $m$  possible outcomes, is a collection of projectors.  $P_1, \dots, P_m$  that all act on the same space and

sum to identity  $\sum_{j=1}^m P_j = I$ . These projectors are *pairwise orthogonal*,  $P_i P_j = \delta_{ij} P_i$  where  $\delta_{ij}$  is the Kronecker delta. The Projector  $P_j$  projects onto some subspace  $V_j$  of the total Hilbert Space  $V$ . Now every state  $|\psi\rangle \in V$  can be decomposed in a unique way as  $|\psi\rangle = \sum_{j=1}^m P_j |\psi\rangle$ . Note that since the projectors are orthogonal, the subspaces  $V_j$  are orthogonal and the projection of the states onto  $V_j$ . When we apply this measurement onto the pure state  $|\psi\rangle$ , then we will get outcome  $j$  with probability  $p(j) = \langle\psi| P_j |\psi\rangle$ . If the outcome  $j$  occurs, then the state of the system immediately after the measurement is  $\frac{P_j |\psi\rangle}{\sqrt{p(j)}}$ . Note that the probabilities sum to 1.

We cannot chose which  $P_j$  will be applied to the state but can only give a probability distribution. However, if the state  $|\psi\rangle$  fully lies within one of the subspaces  $V_j$  then the measurement of the outcome will be  $j$  with certainty. Note that the  $m$  projectors together form one measurement, we don't use the word "measurement" for individual  $P_j$ .

**Example 2.3.3.** A measurement in the computational basis in a  $N$ -dimensional state space is a specific projective measurement with  $m=N$  and  $P_j = |j\rangle \langle j|$ .  $P_j$  projects onto the computation basis state  $|j\rangle$  and the corresponding subspace  $V_j \subset V$  is the 1-dimensional subspace spanned by  $|j\rangle$ . Consider the state  $|\psi\rangle = \sum_{j=0}^{N-1} \alpha_j |j\rangle$ .  $P_j |\psi\rangle = \alpha_j |j\rangle$ , so applying our measurement will give outcome  $j$  with probability  $|\alpha_j|^2$  and the state of the system after the measurement will be  $|j\rangle$ .

**Example 2.3.4.** Note that we can choose any orthonormal basis for the projective measurement. For example, consider a quantum state  $|\psi\rangle$  expressed as superposition of standard orthonormal basis states  $|0\rangle, |1\rangle, \dots, |N-1\rangle$ . We can choose any other orthonormal basis  $B$  of states say  $|\phi_0\rangle, |\phi_1\rangle, \dots, |\phi_{N-1}\rangle$  and consider the projective measurement defined by the projectors  $P_j = |\phi_j\rangle \langle \phi_j|$ . This is called measuring in  $B$  basis. Applying this measurement to the state  $|\psi\rangle$  will give outcome  $j$  with probability  $|\langle \phi_j | \psi \rangle|^2$  and the state of the system after the measurement will be  $|\phi_j\rangle$ . Note that if  $|\psi\rangle$  equals to one the basis vectors  $|\phi_j\rangle$  then the measurement gives outcome  $j$  with probability 1.

**Example 2.3.5.** Note that it is not at all necessary for projectors to be of rank 1. For example, we can consider only two projectors on a  $N$  (even) dimensional space such that  $P_1 = \sum_{j < N/2} |j\rangle \langle j|$  and  $P_2 = \sum_{j \geq N/2} |j\rangle \langle j|$ . say the state of a quantum system is  $|\psi\rangle = \frac{1}{\sqrt{3}} |1\rangle + \sqrt{\frac{2}{3}} |N\rangle$ . Then, it gives outcome 1 with probability  $p(1) = \langle\psi| P_1 |\psi\rangle = \frac{1}{3}$  and outcome 2 with probability  $p(2) = \langle\psi| P_2 |\psi\rangle = \frac{2}{3}$ .

**Observables:** We can write projective measurements with operators  $P_1, \dots, P_m$  and associated distinct outcomes  $\lambda_1, \lambda_2, \dots, \lambda_m \in \mathbb{R}$ , can be written as one Matrix

$M = \sum_{i=1}^m \lambda_i P_i$  which is called an observable. It has an advantage that the expected (average) value of the outcome can be easily calculated as  $\langle \psi | M | \psi \rangle$ . Note that  $M$  is Hermitian. Every observable is a Hermitian.

**Definition 2.3.1.** Projective Measurement: A projective measurement is a measurement described by an observable  $M$ , a Hermitian operator on the state space of the system being observed. Since, it's a Hermitian operator hence has a spectral decomposition as:

$$M = \sum_m m P_m$$

where  $P_m$  is the projection onto the eigen space of  $M$  with eigen value  $m$ . **The possible outcomes of the measurement correspond to the eigenvalues,  $m$ , of the observable.** Upon, measuring the state  $|\psi\rangle$ , the probability of getting result  $m$  is given by:

$$p(m) = \langle \psi | P_m | \psi \rangle$$

Given that the outcome  $m$  occurred, the state of the quantum system immediately after the measurement is:

$$\frac{P_m |\psi\rangle}{\sqrt{p(m)}}$$

**Projective measurement as a special case of Postulate 3:** Suppose measurement operators in Postulate 3 in addition to satisfying the completeness relation  $\sum_m M_m^\dagger M_m = I$  also satisfy the relation that  $M_m$  are orthogonal projectors, that is  $M_m^\dagger = M_m$  hence are Hermitian. Then, the Postulate 3 reduces to a Projective measurement as defined. Calculation of Average values for Projective measurements. By definition, the average value of the measurement is:

$$\begin{aligned} \mathbf{E}(M) &= \sum_m m p(m) \\ &= \sum_m m \langle \psi | P_m | \psi \rangle \\ &= \langle \psi | \left( \sum_m m P_m \right) | \psi \rangle \\ &= \langle \psi | M | \psi \rangle \\ \langle M \rangle &= \langle \psi | M | \psi \rangle \end{aligned}$$

### 2.3.2 Positive Operator Valued Measure (POVM) measurements

For some applications, the post-measurement of the state is of little interest. We only care about the final probability distribution on the m-outcomes, then we can use the most general type of measurement called *positive-operator-valued-measrue (POVM)*. For example, when the system is measured only once, at the end. In such cases, the POVM measurement is more general than the projective measurement.

We specify m positive semi-definite (Psd) matrices  $E_1, \dots, E_m$  such that  $\sum_m E_m = I$ . When measuring a state  $|\psi\rangle$ , the probability of the outcome i is given by  $Tr(E_i |\psi\rangle \langle \psi|) = \langle \psi | E_i | \psi \rangle$ . A projective measurement is a special case of POVm where the measurement elements  $E_i$  are projectors. (That is  $E_i^2 = E_i$  and  $E_i^\dagger = E_i$ ). The POVM elements  $E_i$  are not necessarily orthogonal. *One can show that every POVM can be simulated by a projective measusrement on a slightly larger space that yields the exact probability distribution over measurement outcomes (Neumark's theoerem)*

**Example 2.3.6.** Suppose in a 2-dimensional state space, we know that it is either in state  $|0\rangle$  or  $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ . Clearly, this two states are not orthogonal, so there is no measurement that distinguishes them perfectly. However, there is a POVm measurement that never makes a mistake, but sometimes gives another outcome 2, meaning "I don't know". That is, you would like to do a meaurement with three possible outcomes: 0,1 and 2, such that:

- If the state is  $|0\rangle$ , then you get a correct outcome 0 with proabability 1/4 and outcome 2 with probability 3/4 but never get innncorrect outcome 1.
- If the state is  $|+\rangle$ , then you get correct outcome 1 with probability 1/4 and outcome 2 with probabiltiy 3/4 but never get the innncorrect outcome 0.

This cannot be achieved with projective measurement on the qubit, but can be achieved with follwoing 3-outcome POVM:

- $E_0 = \frac{1}{2} |-\rangle \langle -|$ , (where  $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$  which is orthogonal to the  $|+\rangle$  state.)
- $E_1 = \frac{1}{2} |1\rangle \langle 1|$  (note that this is orthogonal to the sate  $|0\rangle$  state)
- $E_I - E_0 - E_1$

Here  $E_0, E_1, E_2$  are all psd and add up to identity, so they form a valid POVM. None of the three matrices are projectors. (note the success probability can be improved further).

**Definition 2.3.2.** Positive Operator Valued Measure (POVM): Suppose a measurement is described by measurement operator  $M_m$  is performed upon a quantum state  $|\psi\rangle$ . Then the probability of the outcome  $m$  is given by  $p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle$ . Suppose we define  $E_m = M_m^\dagger M_m$ , then clearly  $E_m$  is a positive operator. A POVM measurement is a measurement described by a set of measurement operators  $\{E_m\}$  that satisfy the completeness relation  $\sum_m E_m = I$  and  $p(m) = \langle\psi|E_m|\psi\rangle$ . Thus set of operators  $E_m$  are sufficient to determine the probabilities of the different measurement outcomes. The operators  $E_m$  are known as the POVM elements associated with the measurement. The complete set  $\{E_m\}$  is known as POVM.

## 2.4 Postulate 4: Composite Systems

Consider a composite system made up of two or more distinct quantum physical systems. What are the states of the composite system? How the state space of a composite system is built up from the state spaces of the component systems.

**Definition 2.4.1.** The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Say, if the systems are numbered through 1 to  $n$  and the system number  $i$  is prepared in the state  $|\psi_i\rangle$  then the joint state of the total system is  $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \dots \otimes |\psi_n\rangle$ . This is called the tensor product state of the component states.

**Example 2.4.1.** Consider the orthonormal basis vectors of a single qubit system  $|0\rangle$  and  $|1\rangle$ . Then the orthonormal basis vectors of a two qubit system are:

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

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

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

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

Thus,  $|00\rangle, |01\rangle, |10\rangle, |11\rangle$  are the orthonormal basis vectors of the two qubit Quantum System. Thus, the state of a particle for a two qubit quantum mechanical system can be written as  $|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$  where  $\alpha, \beta, \gamma, \delta$  are complex numbers such that  $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$  and in the matrix form it can be written in matrix form as:

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix}$$

Thus, say if  $|\psi\rangle = \begin{pmatrix} 7 \\ 0 \\ \iota+3 \\ 0 \end{pmatrix}$  then the state of the two qubit system is  $|\psi\rangle = 7|00\rangle + (\iota+3)|10\rangle$ .

This, same concept of representing two-qubit quantum mechanical systems can also be extended to n-qubit quantum mechanical systems. It will be a vector in a  $2^n$ -dimensional complex vector space. For example, in three dimensional vector space there will be  $2^3 = 8$  basis vectors  $|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle$ .  $|101\rangle$  will be represented in matrix form as:

$$|101\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

### 2.4.1 Full and Partial Measurements of a multi-particle systems

A full measurement is a measurement that is performed on all the particles of a multi-particle system. A partial measurement is a measurement that is performed on some of the particles of a multi-particle system. Recall that when we measure a particle in the computational basis it collapses to one of the classical states.

**Example 2.4.2.** Consider a two qubit system in the state  $|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ . A full measurement on the two qubit system will give the outcome 00 with probability  $|\alpha|^2$ , outcome 01 with probability  $|\beta|^2$ , outcome 10 with probability  $|\gamma|^2$  and outcome 11 with probability  $|\delta|^2$ . Thus, also note that the sum of the probabilities of all the outcomes is 1, it is assumed the the state is initially normalized ( $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$ ).

A partial measurement on the two qubit system is a measurement performed on one of the qubits. Say, we perform a measurement on the first qubit, then the state of the second qubit will be the state of the system after the measurement on the first qubit.

Suppose on measurement of the first qubit the probability that it collapsed to state 1 is given by  $|\gamma|^2 + |\delta|^2$  which is probabiltiy that the state collapses to  $|10\rangle$  i.e.  $|\gamma|^2$  plus the probability that the state collapses to the state  $|11\rangle$  i.e.  $|\delta|^2$ , thus probabiltiy of the first qubit collapsing to the state 1 is given as:  $|\gamma|^2 + |\delta|^2$ . Then the state of the second qubit after the the measurement of the first qubit collapsed to 1 will be:

$$\frac{\gamma|10\rangle + \delta|11\rangle}{\sqrt{|\gamma|^2 + |\delta|^2}}$$

here we are normalizing the state in order to satisfy the normalized condition. Similarly, the probabiltiy of the first qubit collapsing to the state 0 is  $|\alpha|^2 + |\beta|^2$  and the state of the second qubit after the measurement of the first qubit collapsed to 0 will be:

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

**Example 2.4.3.** Consider a two qubit system  $|\psi\rangle = \frac{1}{2}|00\rangle - \frac{1}{2}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$ . Then the probability of the first qubit collapsing to the state 0 is  $|\frac{1}{2}|^2 = \frac{1}{4}$  and the probability of the first qubit collapsing to the state 1 is  $|- \frac{1}{2}|^2 + |\frac{1}{\sqrt{2}}|^2 = \frac{3}{4}$ . Thus, the state of the second qubit after the measurement of the first qubit collapsed to 0 will be:

$$\frac{\frac{1}{2}|00\rangle}{\sqrt{|\frac{1}{2}|^2}} = |00\rangle$$

and the state of the second qubit after the measurement of the first qubit collapsed to 1 will be:

$$\frac{-\frac{\iota}{2}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle}{\sqrt{|\frac{\iota}{2}|^2 + |\frac{1}{\sqrt{2}}|^2}} = \frac{-\iota}{\sqrt{3}}|10\rangle + \sqrt{\frac{2}{3}}|11\rangle$$

We can also, ask the question in reverse, that is if we measure the second qubit, then what is the state of the first qubit. The probability of the second qubit collapsing to the state 0 is  $|\frac{1}{2}|^2 + |-\frac{\iota}{2}|^2 = \frac{1}{2}$  and the probability of the second qubit collapsing to the state 1 is  $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$ . Thus, the state of the first qubit after the measurement of the second qubit collapsed to 0 will be:

$$\frac{\frac{1}{2}|00\rangle - \frac{\iota}{2}|10\rangle}{\sqrt{|\frac{1}{2}|^2 + |-\frac{\iota}{2}|^2}} = \frac{1}{\sqrt{2}}|00\rangle - \frac{\iota}{\sqrt{2}}|10\rangle$$

and the state of the first qubit after the measurement of the second qubit collapsed to 1 will be:

$$\frac{\frac{1}{\sqrt{2}}|11\rangle}{\sqrt{|\frac{1}{\sqrt{2}}|^2}} = |11\rangle$$

**Example 2.4.4.** Consider a state  $|\psi\rangle = \frac{1}{\sqrt{5}}|0000\rangle - \sqrt{\frac{2}{5}}|0100\rangle + \sqrt{\frac{1}{5}}|1111\rangle + \sqrt{\frac{1}{5}}|0110\rangle$ . We wish to find the probability and resultant state is the 1st and 4th qubits are 0. The probability of the 1st qubit and 4th qubit both collapsing to 0 is  $|\frac{1}{\sqrt{5}}|^2 + |-\sqrt{\frac{2}{5}}|^2 + |\sqrt{\frac{1}{5}}|^2 = \frac{4}{5}$ . Thus, the resultant state of the 2nd and 3rd qubits after the measurement of the 1st and 4th qubits collapsed to 0 will be:

$$\frac{\frac{1}{\sqrt{5}}|0000\rangle - \sqrt{\frac{2}{5}}|0100\rangle + \sqrt{\frac{1}{5}}|0110\rangle}{\sqrt{\frac{4}{5}}} = \frac{1}{2}|0000\rangle - \frac{1}{\sqrt{2}}|0100\rangle + \frac{1}{2}|0110\rangle$$

# Chapter 3

## Qubits

A qubit is a two-dimensional quantum system. The state of a qubit can be represented as a vector in a two-dimensional complex vector space.

### 3.1 Bloch-Poincare sphere representation of a Qubit

Consider a Qubit in an arbitrary state  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  and  $|0\rangle$  and  $|1\rangle$  are orthonormal basis states,  $\alpha$  and  $\beta$  are complex numbers such that  $|\alpha|^2 + |\beta|^2 = 1$ . How many real parameters are required to specify the state of a qubit?. Now since  $\alpha$  and  $\beta$  are complex numbers, we can write them as  $\alpha = |\alpha|e^{i\phi_\alpha}$  and  $\beta = |\beta|e^{i\phi_\beta}$ . Here,  $|\alpha|$  and  $|\beta|$  are real numbers and  $\phi_\alpha$  and  $\phi_\beta$  are real numbers. Thus, we need 4 real numbers to specify the state of a qubit. Now putting the condition that  $|\alpha|^2 + |\beta|^2 = 1$ , we can write  $|\beta|^2 = 1 - |\alpha|^2 \implies |\beta| = \sqrt{1 - |\alpha|^2}$ . Thus, we can rewrite the state of a qubit as:

$$|\psi\rangle = |\alpha|e^{i\phi_\alpha}|0\rangle + \sqrt{1 - |\alpha|^2}e^{i\phi_\beta}|1\rangle$$

Now the unknowns are  $|\alpha|, \phi_\alpha, \phi_\beta$ . Taking  $\phi_\alpha$  common we get:

$$|\psi\rangle = e^{i\phi_\alpha} \left( |\alpha| |0\rangle + \sqrt{1 - |\alpha|^2} e^{i(\phi_\beta - \phi_\alpha)} |1\rangle \right)$$

Now, we know that the state of a qubit is defined upto a global phase. Thus, ignoring the term  $e^{i\phi_\alpha}$  we can write the state of a qubit as:

$$|\psi\rangle = |\alpha| |0\rangle + \sqrt{1 - |\alpha|^2} e^{i\phi} |1\rangle$$

where  $\phi = \phi_\beta - \phi_\alpha$  is the relative phase between the states  $|0\rangle$  and  $|1\rangle$ . Now, on substituting  $|\alpha| = \cos(\theta/2)$ ,  $\sqrt{1 - |\alpha|^2} = \sin(\theta/2)$ , we get:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle$$

This cannot be further simplified to reduce the number of variables. Thus, we can clearly see that we need two real numbers -  $\theta$  and  $\phi$  to specify the state of a qubit. The state of a qubit can be represented as a point on the surface of a unit sphere in a three-dimensional space. This sphere is called the Bloch Sphere. A visual representation of the Bloch Sphere (or Bloch-Poincare Sphere) is shown in the figure 3.1.

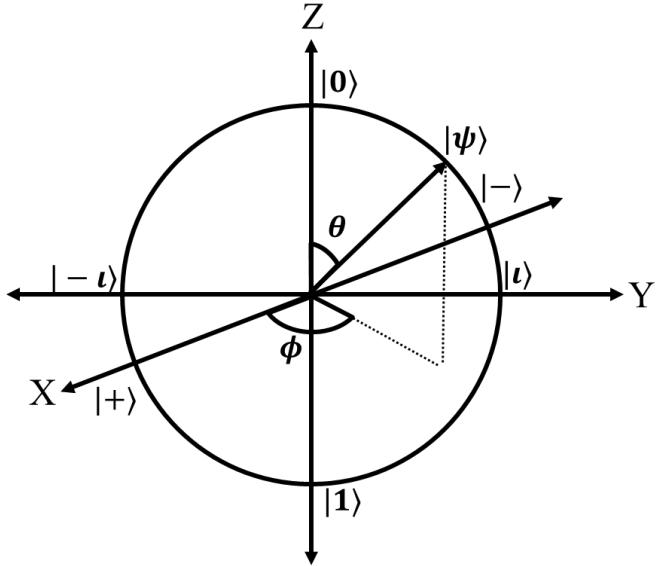


Figure 3.1: Bloch-Poincare Sphere

Thus for different values of  $\theta$  and  $\phi$  we get different points on the bloch sphere which represents the different states of a qubit. For a unique mapping from the  $\theta$  and  $\phi$  to the points on the Bloch Sphere, we restrict the value of  $\theta \in [0, \pi]$  and  $\phi \in [0, 2\pi]$ . Thus, the state of a qubit can be represented as:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle$$

where  $\theta \in [0, \pi]$  and  $\phi \in [0, 2\pi]$  are the polar (represents magnitude or argument) and azimuthal (represents relative phase) angles of the Bloch Sphere respectively.

The following code using Qiskit can be used to plot the Bloch Sphere. For running the code you need to install the qiskit library. Click here for guide on installing qiskit

```

1 import matplotlib.pyplot as plt
2 from qiskit.visualization import plot_bloch_vector # Import the
   function to plot the Bloch vector
3 from qiskit.visualization import plot_bloch_multivector # Import the
   function to plot the Bloch vector
4
5 """
6 Function to plot the Bloch Sphere
7
8 plot_bloch_vector(bloch, title='', ax=None, figsize=None, coord_type='
   spherical', font_size=None)
9
10 Plots a Bloch Sphere
11
12 Parameters
13 - bloch (list [double]) – array of three elements where [ $x$ ,  $y$ ,  $z$ ] (Cartesian) or
14 [ $r$ ,  $\theta$ ,  $\phi$ ] (spherical in radians)  $\theta$  is inclination
   angle from +z direction
15  $\phi$  is azimuth from +x direction
16 - title (str) – a string that represents the title of the plot
17 - ax (matplotlib.axes.Axes) – an axes of the current figure to plot the
   Bloch sphere into
18 - figsize (tuple) – a tuple (width, height) in inches that represents
   the size of the figure
19 - coord_type (str) – the coordinate system to use. 'spherical' or '
   cartesian', default is cartesian
20 - font_size (int) – the font size of the tex
21
22 Returns
23 - matplotlib.figure.Figure – a matplotlib figure object
24
25 Raises
26 - ImportError – if matplotlib is not installed
27 """
28
29 plot_bloch_vector([0,1,0], title='Bloch Sphere')

```

The output of the above code is shown in the figure 3.2.

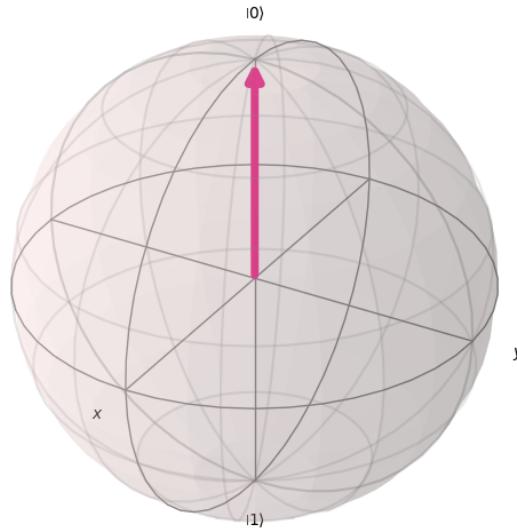


Figure 3.2: Bloch Sphere

Note that the Bloch Sphere is a geometric representation of the pure states of a qubit. The mixed states of a qubit are represented by the interior of the Bloch Sphere.

**Example 3.1.1.** The state of a qubit for different values of  $\theta$  and  $\phi$  is shown in the table 3.1.

$\theta$	$\phi$	State of Qubit	At Axis
0	0	$ 0\rangle$	+ve Z
$\pi$	0	$ 1\rangle$	-ve Z
$\frac{\pi}{2}$	0	$ +\rangle = \frac{1}{\sqrt{2}}( 0\rangle +  1\rangle)$	+ve X
$\frac{\pi}{2}$	$\pi$	$ -\rangle = \frac{1}{\sqrt{2}}( 0\rangle -  1\rangle)$	-ve X
$\frac{\pi}{2}$	$\frac{\pi}{2}$	$ +i\rangle = \frac{1}{\sqrt{2}}( 0\rangle + i 1\rangle)$	+ve Y
$\frac{\pi}{2}$	$-\frac{\pi}{2}$	$  -i\rangle = \frac{1}{\sqrt{2}}( 0\rangle - i 1\rangle)$	-ve Y

Table 3.1: State of a Qubit for different values of  $\theta$  and  $\phi$ 

**Quantum Gates acting on Qubits can be thought of as rotations on the Bloch Sphere.** This will be explained further in the later sections.

Note that orthonormal states reside at the ends of the diameter of the Bloch Sphere. Let a state  $|\psi\rangle$  be represented by a point on the Bloch Sphere. The state

$|\psi\rangle$  can be written as:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle$$

Then the state (say  $|\psi'\rangle$ ) exactly opposite on the Bloch sphere will have the state:

$$|\psi'\rangle = \cos\left(\frac{\pi+\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\pi+\theta}{2}\right)|1\rangle$$

Thus, upon simplifying we get:

$$|\psi'\rangle = \cos\left(\frac{\pi+\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\pi+\theta}{2}\right)|1\rangle$$

$$|\psi'\rangle = -\sin\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\cos\left(\frac{\theta}{2}\right)|1\rangle$$

Now taking the inner product of the states  $|\psi\rangle$  and  $|\psi'\rangle$  we get:

$$\langle\psi|\psi'\rangle = -\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\theta}{2}\right) + e^{i\phi}e^{-i\phi}\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\theta}{2}\right) = 0$$

Thus, the inner product is zero which implies that the states on the opposite ends of the Bloch Sphere are orthogonal.

**Note:** Such a correspondence does not exist for higher dimensions i.e. No such visualization exists for say Qudits which is d level quantum systems. From figure 3.1 we can see that to convert from polar to cartesian coordinates we use the following relations:

$$x = r \sin \theta \cos \phi$$

$$y = r \sin \theta \sin \phi$$

$$z = r \cos \theta$$

note that here  $r = 1$  (from pure states i.e. points on the surface of the sphere). To convert from cartesian to polar coordinates we use the following relations:

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\theta = \cos^{-1}(z)$$

$$\phi = \tan^{-1}\left(\frac{y}{x}\right)$$

Let us denote the vector as  $\vec{n} = (\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta, \cos \theta)$ . Points on a Bloch sphere is always a pure state.

## 3.2 Single Qubit Gates

As said earlier that quantum gates are unitary operators that act on the state of a quantum system. Recall from Appendix that the unitary operators are the ones that preserve the inner product of the vectors and the norm of the vectors i.e. they only rotate the vectors in the complex vector space (Rotation matrices in  $\mathbb{C}^n$ ). Thus, a state normalized initially remains in the normalized state and does not change its norm upon the action of Unitary gates/operators. The single qubit gates are the quantum gates that act on a single qubit, represented by  $2 \times 2$  matrices. Since a quantum state is represented by a column vector (in some basis, generally standard basis). A gate is a unitary matrix/transformation that acts on the qubit(column vector) and changes its state to some other state. They are rotations on the Bloch Sphere.

**Example 3.2.1.** Consider a  $2 \times 2$  matrix  $U$  given as:

$$U = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

in the orthonormal input and output basis  $\{|0\rangle, |1\rangle\}$ , thus, we can write the matrix  $U$  as:

$$U = a|0\rangle\langle 0| + b|0\rangle\langle 1| + c|1\rangle\langle 0| + d|1\rangle\langle 1|$$

This, will be used to express the Pauli Matrices in the later sections.

### 3.2.1 Pauli Matrices

#### Pauli-I Gate

It is used for the measurement of decoherence (Qubits start to lose its Quantum abilities after certain time). The Truth table is given as in table 3.2. It does no rotation of the bloch vector on the Bloch Sphere.

Input	Output
$ 0\rangle$	$ 0\rangle$
$ 1\rangle$	$ 1\rangle$

Table 3.2: Truth Table for Pauli-I Gate

The code for the Pauli-I Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5
6 circuit = QuantumCircuit(qreg_q)
7
8 circuit.id(qreg_q[0]) #For Pauli-Identity Gate
9 circuit.draw(output='mpl')#.savefig ('../images/pauli-i.png') #Draw the
  circuit

```

The circuit symbol is as shown in figure 3.3



Figure 3.3: Pauli-I Gate

In operator Notation it is:

$$\begin{aligned} I |0\rangle &= |0\rangle \\ I |1\rangle &= |1\rangle \end{aligned}$$

It is represented by the matrix in the computational basis ( $|0\rangle, |1\rangle$ ) as:

$$\sigma_I = I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

It's Eigen values are 1 and 1. It's Eigen vectors are any vector in the 2-D Space. Note that it's Unitary( $II^\dagger = I^\dagger I = I; I = I^{-1}$ ) and Hermitian Matrix( $I = I^\dagger$ ), thus a Normal Matrix ( $II^\dagger = I^\dagger I$ ) hence Unitarily digonalizable. Thus, it has a Spectral decomposition (outer product representation) which can be written as:

$$I = |0\rangle\langle 0| + |1\rangle\langle 1|$$

Action of Pauli-I Gate on a general qubit  $\alpha|0\rangle + \beta|1\rangle$  is:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \alpha|0\rangle + \beta|1\rangle$$

### Pauli-X Gate

It is also known as the NOT gate. It is equivalent of Classical NOT Gate. It is thus called Bit Flip Gate. The truth table is as in 3.3. It does anticlockwise  $\pi$  rotation about X axis of Bloch sphere. It flips the state  $|0\rangle$  to  $|1\rangle$  and  $|1\rangle$  to  $|0\rangle$ .

Input	Output
$ 0\rangle$	$ 1\rangle$
$ 1\rangle$	$ 0\rangle$

Table 3.3: Truth Table for Pauli-X Gate

The code for the Pauli-X Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5 circuit = QuantumCircuit(qreg_q, name='Not Gate')
6
7 circuit.x(qreg_q[0])
8
9 circuit.draw(output='mpl')#.savefig ('../images/pauli-x.png') #Draw the
    circuit

```

The circuit symbol is as shown in figure 3.4.



Figure 3.4: Pauli-X Gate

In the operator Notation it is:

$$\begin{aligned} X |0\rangle &= |1\rangle \\ X |1\rangle &= |0\rangle \end{aligned}$$

It is represented by the matrix in the computational basis ( $|0\rangle, |1\rangle$ ) as:

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

It's Eigen values are 1 and -1. It's corresponding Eigen vectors are  $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$  and  $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ . It is a Unitary and Hermitian Matrix. It is a Normal Matrix hence Unitarily Diagonalizable. It has a Spectral decomposition (outer product representation) which can be written as:

$$X = |+\rangle \langle +| - |-\rangle \langle -|$$

Upon, substituting the values of  $|+\rangle$  and  $|-\rangle$  we get:

$$\begin{aligned} X &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \frac{1}{\sqrt{2}}(\langle 0| + \langle 1|) - \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \frac{1}{\sqrt{2}}(\langle 0| - \langle 1|) \\ &= \frac{1}{2}(|0\rangle \langle 0| + |0\rangle \langle 1| + |1\rangle \langle 0| + |1\rangle \langle 1|) + \frac{1}{2}(|0\rangle \langle 0| - |0\rangle \langle 1| - |1\rangle \langle 0| + |1\rangle \langle 1|) \\ &= |0\rangle \langle 1| + |1\rangle \langle 0| \end{aligned}$$

Thus, the Pauli-X Gate can also be written as:

$$X = |0\rangle \langle 1| + |1\rangle \langle 0|$$

**Note that the Pauli-X Gate can be written as  $X = HZH$  or  $\sigma_X = H\sigma_ZH$  where H and Z are the Hadamard and Pauli-Z Gates respectively.** Action of Pauli-X Gate on a general qubit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  is:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} = \beta|0\rangle + \alpha|1\rangle$$

### Pauli-Y Gate

It is a anticlockwise  $\pi$  rotation about Y axis of Bloch Sphere. It flips the Qubit and multiplies a complex amplitude. The truth table is as in table 3.4.

Input	Output
$ 0\rangle$	$i 1\rangle$
$ 1\rangle$	$-i 0\rangle$

Table 3.4: Truth Table for Pauli-Y Gate

The code for the Pauli-Y Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5 circuit = QuantumCircuit(qreg_q, name='Not Gate')
6
7 circuit.y(qreg_q[0]) #Apply the Y gate
8
9 circuit.draw(output='mpl')#.savefig ('../images/pauli-x.png') #Draw the
  circuit

```

The circuit symbol is as shown in figure 3.5.



Figure 3.5: Pauli-Y Gate

In the operator Notation it is:

$$\begin{aligned} Y|0\rangle &= i|1\rangle \\ Y|1\rangle &= -i|0\rangle \end{aligned}$$

It is represented by the matrix in the computational basis ( $|0\rangle, |1\rangle$ ) as:

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

It's Eigen values are 1 and -1. It's corresponding Eigen vectors are  $|+i\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$  and  $|-i\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$ . It is a Unitary and Hermitian Matrix. It is a Normal Matrix hence Unitarily Diagonalizable. It has a Spectral decomposition (outer product representation) which can be written as:

$$Y = |+i\rangle\langle+i| - |-i\rangle\langle-i|$$

Upon, substituting the values of  $|+i\rangle$  and  $|-i\rangle$  we get:

$$\begin{aligned} Y &= \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)\frac{1}{\sqrt{2}}(\langle 0| - i\langle 1|) - \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)\frac{1}{\sqrt{2}}(\langle 0| + i\langle 1|) \\ &= \frac{1}{2}(|0\rangle\langle 0| + i|0\rangle\langle 1| - i|1\rangle\langle 0| + i|1\rangle\langle 1|) - \frac{1}{2}(|0\rangle\langle 0| - i|0\rangle\langle 1| + i|1\rangle\langle 0| + i|1\rangle\langle 1|) \\ &= i|0\rangle\langle 1| - i|1\rangle\langle 0| \end{aligned}$$

Thus, the Pauli-Y Gate can also be written as:

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

**Note that the Pauli-Y Gate can be written as  $Y = \iota XZ$  or  $\sigma_Y = \iota \sigma_X \sigma_Z$  where  $X$  and  $Z$  are the Pauli-X and Pauli-Z Gates respectively.** This means that the Pauli-Y Gate is a rotation about the Y-axis of the Bloch Sphere by  $\pi$  in anticlockwise direction and is equivalent to a rotation about the X-axis by anticlockwise  $\pi$  followed by a rotation about the Z-axis by anticlockwise  $\pi$  (upto a global phase).

Action of Pauli-Y Gate on a general qubit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  is:

$$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} -i\beta \\ i\alpha \end{bmatrix} = i\alpha|1\rangle - i\beta|0\rangle$$

### Pauli-Z Gate

It is a anticlockwise  $\pi$  rotation about Z axis of Bloch Sphere. It flips the sign of the  $|1\rangle$  state. It is also called Phase Flip/Phase shoft/Sign flip gate. The truth table is as in table 3.5.

Input	Output
$ 0\rangle$	$ 0\rangle$
$ 1\rangle$	$- 1\rangle$

Table 3.5: Truth Table for Pauli-Z Gate

The code for the Pauli-Z Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5 circuit = QuantumCircuit(qreg_q, name='Not Gate')
6
7 circuit.z(qreg_q[0]) #Apply the Z gate
8
9 circuit.draw(output='mpl')#.savefig ('../images/pauli-z.png') #Draw the
  circuit

```

The circuit symbol is as shown in figure 3.6.



Figure 3.6: Pauli-Z Gate

In the operator Notation it is:

$$\begin{aligned} Z |0\rangle &= |0\rangle \\ Z |1\rangle &= -|1\rangle \end{aligned}$$

It is represented by the matrix in the computational basis  $(|0\rangle, |1\rangle)$  as:

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

It's Eigen values are 1 and -1. It's corresponding Eigen vectors are  $|0\rangle$  and  $|1\rangle$ . It is a Unitary and Hermitian Matrix. It is a Normal Matrix hence Unitarily Diagonalizable. It has a Spectral decomposition (outer product representation) which can be written as:

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

In Compact form, we can write Pauli-Z Gate as:

$$Z |j\rangle = (-1)^j |j\rangle$$

where  $j = 0, 1$ . Action of Pauli-Z Gate on a general qubit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  is:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \\ -\beta \end{bmatrix} = \alpha|0\rangle - \beta|1\rangle$$

### Important Note

Note that all the Pauli-gate are Unitary and Hermitian. Since, all the Unitary gates ( $U^\dagger = U^{-1}$ ) are reversible. Thus, from Postulate 2: Evolution, we can apply Pauli Gates. All the Pauli gates are reversible and thus can be applied in Quantum Circuits. The four Pauli gates can also be denoted as  $X = \sigma_X$ ,  $Y = \sigma_Y$ ,  $Z = \sigma_Z$  and  $I = \sigma_I$ . All the Pauli Gates as said earlier are anticlockwise rotations on the Bloch Sphere. The Pauli-X Gate is a  $\pi$  rotation about the X-axis, Pauli-Y Gate is a  $\pi$  rotation about the Y-axis and Pauli-Z Gate is a  $\pi$  rotation about the Z-axis. The Pauli-I Gate does not do any rotation on the Bloch Sphere. Also note that applying the same gate twice will be equivalent to applying the Identity Gate. Thus, the Pauli Gates are self-inverse. Since they are hermitian hence,  $U^\dagger = U$  and they are unitary thus,  $U^\dagger U = UU^\dagger = I$ . Thus combining, the two we get  $U^2 = I$ . This, can also be verified as shown below:

$$X^2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

$$Y^2 = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

$$Z^2 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

$$I^2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

Thus, we can see that all the Pauli Gates are self-inverse. The Pauli gates also form the basis for the single qubit gates. Any single qubit gate can be written as a linear combination of the Pauli Gates. In other words, any  $2 \times 2$  matrix can be written as a linear combination of the Pauli Matrices.

$$A = \alpha I + \beta X + \gamma Y + \delta Z$$

where  $\alpha, \beta, \gamma, \delta$  are complex numbers. The Pauli Matrices, except Identity are also traceless (trace=0) and all the Pauli matrices are Hermitian. Note here it should be presumed that the rotations are always in anticlockwise direction.

### 3.2.2 Hadamard Gate

It creates a superposition of Qubits. It is a  $\pi$  rotation about the X+Z axis of Bloch Sphere or a  $\pi/2$  rotation about the Y axis followed by a  $\pi$  rotation about the X axis. It is also called as the Square root of NOT gate. The truth table is as in table 3.6.

Input	Output
$ 0\rangle$	$ +\rangle = \frac{1}{\sqrt{2}}( 0\rangle +  1\rangle)$
$ 1\rangle$	$ -\rangle = \frac{1}{\sqrt{2}}( 0\rangle -  1\rangle)$

Table 3.6: Truth Table for Hadamard Gate

The code for the Hadamard Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5 circuit = QuantumCircuit(qreg_q, name='Hadamard Gate')
6
7 circuit.h(qreg_q[0]) #Apply the H gate
8
9 circuit.draw(output='mpl')#.savefig ('../images/hadamard.png') #Draw the
  circuit

```

The circuit symbol is as shown in figure 3.7.

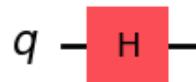


Figure 3.7: Hadamard Gate

In the operator Notation it is:

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

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

Generally in a quantum circuit we initialise a qubit in state  $|0\rangle$  and thus Hadamard gate can be used to create a superposition. It is represented by the matrix in the computational basis ( $|0\rangle, |1\rangle$ ) as:

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

In compact form we can write the action of Hadamard Gate on one qubit as:

$$H|j\rangle = \frac{1}{\sqrt{2}} \sum_{k=0}^1 (-1)^{jk} |k\rangle$$

where  $j = 0, 1$ . It is a Unitary and Hermitian Matrix. It is a Normal Matrix hence Unitarily Diagonalizable. It has a Spectral decomposition (outer product representation) which can be written as:

$$H = |+\rangle\langle 0| + |-\rangle\langle 1|$$

Action of Hadamard Gate on a general qubit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  is:

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{\alpha + \beta}{\sqrt{2}} \\ \frac{\alpha - \beta}{\sqrt{2}} \end{bmatrix} = \frac{\alpha + \beta}{\sqrt{2}} |0\rangle + \frac{\alpha - \beta}{\sqrt{2}} |1\rangle$$

Note that the Hadamard gate is also a self-inverse gate. Since its Hermitian as well as Unitary, thus  $H^\dagger = H$  and  $H^\dagger H = HH^\dagger = I$ ;  $H = H^{-1}$ . Thus,  $H^2 = I$ . Intuitively this means that applying Hadamard gate twice will bring the quantum state back to the same quantum state. This can also be imagined as doing rotations on bloch sphere and then arriving back at the same state.

$$H^2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1+1 & 1-1 \\ 1-1 & 1+1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

Thus, we can see that the Hadamard gate is self-inverse. This means that if I apply Hadamard gate twice on  $|0\rangle$  or on  $|1\rangle$  then we will arrive back at the same state.

$$H^2|0\rangle = H|+\rangle = |0\rangle$$

$$H^2|1\rangle = H|-\rangle = |1\rangle$$

### Hadamard Gates on Multiple Qubits

Hadamard gate can also be applied on multiple qubits. The Hadamard gate on multiple qubits (say n qubits) using tensor products is given as:

$$H^{\otimes n} = \underbrace{H \otimes H \otimes \dots \otimes H}_{n \text{ times}}$$

One of the way to find the Hadamard matrix is to perform the tensor product n times to get the Hadamard matrix acting on the n qubits. For example, we can find Hadamard gate acting on 2 qubits in matrix form using tensor product as:

$$H^{\otimes 2} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Clearly this does not scale. Another method is, we can find the action of Hadamard gate on multiple qubits by applying the Hadamard gate on each qubit separately.

$$H^{\otimes n}(|0\rangle \otimes |0\rangle \otimes \dots \otimes |0\rangle) = H|0\rangle \otimes H|0\rangle \otimes \dots \otimes H|0\rangle = |+\rangle \otimes |+\rangle \otimes \dots \otimes |+\rangle$$

This is a better method then writing entire matrix for n qubits. Thus, we can see that the Hadamard gate on multiple qubits creates a superposition of all possible states of n qubits. So, for writing the Hadamard gate on multiple qubits, in a compact form, first we consider the compact form of  $H^{\otimes 2}$  as:

$$H^{\otimes 2}|x\rangle = H|x_1\rangle \otimes H|x_2\rangle$$

where  $x = x_1x_2$ . Now using the fact that  $H|x_1\rangle = \frac{1}{\sqrt{2}} \sum_{y_1 \in \{0,1\}} (-1)^{x_1y_1} |y_1\rangle$  and  $H|x_2\rangle = \frac{1}{\sqrt{2}} \sum_{y_2 \in \{0,1\}} (-1)^{x_2y_2} |y_2\rangle$ , we get:

$$H^{\otimes 2}|x\rangle = \frac{1}{2} \sum_{y_1, y_2 \in \{0,1\}} (-1)^{x_1y_1} (-1)^{x_2y_2} |y_1\rangle \otimes |y_2\rangle = \frac{1}{2} \sum_{y \in \{0,1\}^2} (-1)^{x \cdot y} |y\rangle$$

where  $x = x_1x_2$  and  $y = y_1y_2$ . Thus, it can be generalized and we can write the Hadamard gate on n qubits as:

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle$$

where  $x = x_1x_2 \dots x_n$  and  $y = y_1y_2 \dots y_n$  and  $x \cdot y$  is the dot product between  $x$  and  $y$ . Thus, we can see that the Hadamard gate on  $n$  qubits creates a superposition of all possible states of  $n$  qubits. In general, we start with state  $|0\rangle$  for all the qubits initially, thus Action of hadamard gate on  $n$  qubits all in state  $|0\rangle$  is:

$$H^{\otimes n} |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} |y\rangle$$

#### Example 3.2.2. Fast Hadamard Gate using Tabular method :

Consider a state  $|\psi\rangle = \frac{|000\rangle + |110\rangle - |100\rangle - |111\rangle}{\sqrt{4}}$ . We are suppose to find  $H^{\otimes 3} |\psi\rangle$ . Thus, on solving using the equation for hadamard gate acting on multiple qubits  $H^{\otimes n} |x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle$  we get,

$$\frac{1}{\sqrt{8}} (|001\rangle - |011\rangle + |100\rangle + |110\rangle + |111\rangle)$$

#### 3.2.3 $S$ and $S^\dagger$ Gate

These are some specialised rotations,

##### $S$ Gate

It is  $\pi/2$  rotation around Z axis of Bloch Sphere. It is also called as the Phase Gate. The truth table is as in table 3.7.

Input	Output
$ 0\rangle$	$ 0\rangle$
$ 1\rangle$	$i 1\rangle$

Table 3.7: Truth Table for S Gate

The code for the S Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5 circuit = QuantumCircuit(qreg_q, name='S Gate')
6
7 circuit.s(qreg_q[0]) #Apply the S gate

```

```

8 circuit.draw(output='mpl')#.savefig( '../images/s-gate.png' ) #Draw the
9 circuit

```

The circuit symbol is as shown in figure 3.8.

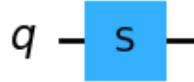


Figure 3.8: S Gate

In the operator Notation it is:

$$\begin{aligned} S |0\rangle &= |0\rangle \\ S |1\rangle &= i |1\rangle \end{aligned}$$

It is represented by the matrix in the computational basis ( $|0\rangle, |1\rangle$ ) as:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = e^{i\pi/4} \begin{bmatrix} e^{-i\pi/4} & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$$

Thus, it is called as  $\pi/4$  gate. It's Eigen values are 1 and  $i$ . It's corresponding Eigen vectors are  $|0\rangle$  and  $|1\rangle$ . It is a Unitary and Hermitian Matrix. It is a Normal Matrix hence Unitarily Diagonalizable. It has a Spectral decomposition (outer product representation) which can be written as:

$$S = |0\rangle \langle 0| + i |1\rangle \langle 1|$$

Action of S Gate on a general qubit  $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$  is:

$$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \\ i\beta \end{bmatrix} = \alpha |0\rangle + i\beta |1\rangle$$

### $S^\dagger$ Gate

It is  $\pi/2$  rotation around Z axis of Bloch Sphere in the opposite direction. It is also called as the Conjugate Phase Gate. The truth table is as in table 3.8.

Input	Output
$ 0\rangle$	$ 0\rangle$
$ 1\rangle$	$-i 1\rangle$

Table 3.8: Truth Table for  $S^\dagger$  Gate

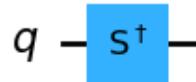
The code for the  $S^\dagger$  Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5 circuit = QuantumCircuit(qreg_q, name='T Gate')
6
7 circuit.sdg(qreg_q[0]) #Apply the sdag gate
8
9 circuit.draw(output='mpl')#.savefig ('../images/sdag-gate.png') #Draw
    the circuit

```

The circuit symbol is as shown in figure 3.9.

Figure 3.9:  $S^\dagger$  Gate

In the operator Notation it is:

$$\begin{aligned} S^\dagger |0\rangle &= |0\rangle \\ S^\dagger |1\rangle &= -i|1\rangle \end{aligned}$$

It is represented by the matrix in the computational basis ( $|0\rangle, |1\rangle$ ) as:

$$S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} = e^{-i\pi/4} \begin{bmatrix} e^{i\pi/4} & 0 \\ 0 & e^{-i\pi/4} \end{bmatrix}$$

It's Eigen values are 1 and  $-i$ . It's corresponding Eigen vectors are  $|0\rangle$  and  $|1\rangle$ . It is a Unitary and Hermitian Matrix. It is a Normal Matrix hence Unitarily Diagonalizable.

It has a Spectral decomposition (outer product representation) which can be written as:

$$S^\dagger = |0\rangle\langle 0| - i|1\rangle\langle 1|$$

Action of  $S^\dagger$  Gate on a general qubit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  is:

$$\begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \\ -i\beta \end{bmatrix} = \alpha|0\rangle - i\beta|1\rangle$$

Note that both  $S$  and  $S^\dagger$  gates are Unitary but not Hermitian thus they are not self-inverse. Recall, that the only criteria for a gate is to be a Unitary (from Postulate 2: Unitary evolution) gate and not necessarily Hermitian.

### 3.2.4 $T$ and $T^\dagger$ Gate

These are some specialised rotations,

#### $T$ Gate

It is  $\pi/4$  rotation around Z axis of Bloch Sphere. It is also called as the  $\pi/8$  Gate. The truth table is as in table 3.9.

Input	Output
$ 0\rangle$	$ 0\rangle$
$ 1\rangle$	$e^{i\pi/4} 1\rangle$

Table 3.9: Truth Table for T Gate

The code for the T Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5 circuit = QuantumCircuit(qreg_q, name='T Gate')
6
7 circuit.sdg(qreg_q[0]) #Apply the T gate
8
9 circuit.draw(output='mpl')#.savefig ('../images/sdag-gate.png') #Draw
    the circuit

```

The circuit symbol is as shown in figure 3.10.



Figure 3.10: T Gate

In the operator Notation it is:

$$\begin{aligned} T |0\rangle &= |0\rangle \\ T |1\rangle &= e^{i\pi/4} |1\rangle \end{aligned}$$

It is represented by the matrix in the computational basis ( $|0\rangle, |1\rangle$ ) as:

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

Thus, it is called as  $\pi/8$  gate. Its Eigen values are 1 and  $e^{i\pi/4}$ . Its corresponding Eigen vectors are  $|0\rangle$  and  $|1\rangle$ . It is a Unitary and Hermitian Matrix. It is a Normal Matrix hence Unitarily Diagonalizable. It has a Spectral decomposition (outer product representation) which can be written as:

$$T = |0\rangle \langle 0| + e^{i\pi/4} |1\rangle \langle 1|$$

Action of T Gate on a general qubit  $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$  is:

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \\ e^{i\pi/4}\beta \end{bmatrix} = \alpha |0\rangle + e^{i\pi/4}\beta |1\rangle$$

### $T^\dagger$ Gate

It is  $\pi/4$  rotation around Z axis of Bloch Sphere in the opposite direction. It is also called as the  $\pi/8$  Gate. The truth table is as in table 3.10.

Input	Output
$ 0\rangle$	$ 0\rangle$
$ 1\rangle$	$e^{-i\pi/4}  1\rangle$

Table 3.10: Truth Table for  $T^\dagger$  Gate

The code for the  $T^\dagger$  Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5 circuit = QuantumCircuit(qreg_q, name='T Gate')
6
7 circuit.tdg(qreg_q[0]) #Apply the T gate
8
9 circuit.draw(output='mpl')#.savefig ('../images/tdag-gate.png') #Draw
    the circuit

```

The circuit symbol is as shown in figure 3.11.

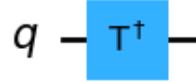


Figure 3.11:  $T^\dagger$  Gate

In the operator Notation it is:

$$\begin{aligned} T^\dagger |0\rangle &= |0\rangle \\ T^\dagger |1\rangle &= e^{-i\pi/4} |1\rangle \end{aligned}$$

It is represented by the matrix in the computational basis ( $|0\rangle, |1\rangle$ ) as:

$$T^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{bmatrix} = e^{-i\pi/8} \begin{bmatrix} e^{i\pi/8} & 0 \\ 0 & e^{-i\pi/8} \end{bmatrix}$$

It's Eigen values are 1 and  $e^{-i\pi/4}$ . It's corresponding Eigen vectors are  $|0\rangle$  and  $|1\rangle$ . It is a Unitary and Hermitian Matrix. It is a Normal Matrix hence Unitarily Diagonalizable. It has a Spectral decomposition (outer product representation) which can be written as:

$$T^\dagger = |0\rangle \langle 0| + e^{-i\pi/4} |1\rangle \langle 1|$$

Action of  $T^\dagger$  Gate on a general qubit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  is:

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \\ e^{-i\pi/4}\beta \end{bmatrix} = \alpha|0\rangle + e^{-i\pi/4}\beta|1\rangle$$

Some relations between S and T Gates:

$$\begin{aligned} S &= T^2 \\ S^\dagger &= T^\dagger T^\dagger \end{aligned}$$

This can be verified as follows. Intuitively, one can think as twice  $\pi/4$  rotation around Z axis is  $\pi/2$  rotation around Z axis. Thus, the relations.

$$\begin{aligned} T^2 &= \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = S \\ T^\dagger T^\dagger &= \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} = S^\dagger \end{aligned}$$

### 3.2.5 Quantum Rotation Gates

These are generalized rotations in the Bloch Sphere denoted as  $R_X, R_Y, R_Z$  about the X, Y and Z axis of Bloch Sphere respectively.

#### $R_X$ Gate

It is a rotation about X axis of Bloch Sphere by an angle  $\theta$ . This is denoted as  $R_\theta^X$ . The matrix representation of  $R_\theta^X$  is:

$$R_\theta^X = \begin{bmatrix} \cos(\theta/2) & -i \sin(\theta/2) \\ -i \sin(\theta/2) & \cos(\theta/2) \end{bmatrix} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} \sigma_X = e^{-i \frac{\theta}{2} \sigma_X}$$

The code for the  $R_X$  Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5 circuit = QuantumCircuit(qreg_q, name='U1 Gate')
6
7 circuit.rx(pi/2, qreg_q[0]) #Apply the RX gate
8
9 circuit.draw(output='mpl')#.savefig ('../images/rx-gate.png') #Draw the
circuit

```

The circuit symbol is as shown in figure 3.12.

Figure 3.12:  $R_X$  Gate

Note that the value  $\pi/2$  given in the figure 3.12 denotes the angle by which we rotate around the X Axis of Bloch Sphere.

### $R_Y$ Gate

It is a rotation about Y axis of Bloch Sphere by an angle  $\theta$ . This is denoted as  $R_\theta^Y$ . The matrix representation of  $R_\theta^Y$  is:

$$R_\theta^Y = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix} = \cos \frac{\theta}{2} I - i \cos \frac{\theta}{2} \sigma_Y = e^{-i \frac{\theta}{2} \sigma_Y}$$

The code for the  $R_Y$  Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5 circuit = QuantumCircuit(qreg_q, name='RY Gate')
6
7 circuit.ry(pi/2, qreg_q[0]) #Apply the RY gate
8
9 circuit.draw(output='mpl')#.savefig ('../images/ry-gate.png') #Draw the
  circuit

```

The circuit symbol is as shown in figure 3.13.

Figure 3.13:  $R_Y$  Gate

Note here the value  $\pi/2$  denotes the angle by which we rotate around the Y Axis of Bloch Sphere.

### $R_Z$ Gate

It is a rotation about Z axis of Bloch Sphere by an angle  $\theta$ . This is denoted as  $R_\theta^Z$ . The matrix representation of  $R_\theta^Z$  is:

$$R_\theta^Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} = \cos \frac{\theta}{2} I - i \cos \frac{\theta}{2} \sigma_Z = e^{-i\frac{\theta}{2}\sigma_Z}$$

The code for the  $R_Z$  Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5 circuit = QuantumCircuit(qreg_q, name='RZ Gate')
6
7 circuit.rz(pi/2, qreg_q[0]) #Apply the RZ gate
8
9 circuit.draw(output='mpl')#.savefig ('../images/rz-gate.png') #Draw the
  circuit

```

The circuit symbol is as shown in figure 3.14.

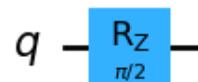


Figure 3.14:  $R_Z$  Gate

Note here the value  $\pi/2$  denotes the angle by which we rotate around the Z Axis of Bloch Sphere.

This Quantum Rotation Gates can be used in making variational circuit as angles can vary.

### 3.2.6 Universal Quantum Gates

To construct any single qubit Quantum gates.

$$U_3(\theta, \phi, \lambda) = \begin{bmatrix} \cos(\theta/2) & -e^{i\lambda} \sin(\theta/2) \\ e^{i\phi} \sin(\theta/2) & e^{i(\phi+\lambda)} \cos(\theta/2) \end{bmatrix}$$

where  $\theta, \phi, \lambda \in \mathbb{R}$  are rotations around x, y and z axis respectively. Here,  $0 \leq \theta \leq \pi$ ,  $0 \leq \phi \leq 2\pi$  and  $0 \leq \lambda \leq 2\pi$ . Similarly, we can fix  $\theta$  and write  $U_2$  gate as:

$$U_2(\phi, \lambda) = \begin{bmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\phi+\lambda)} \end{bmatrix}$$

Here  $\theta = \pi/2$  by default. Similarly, we can fix  $\theta$  and  $\phi$  and write  $U_1$  gate as:

$$U_1(\lambda) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{bmatrix}$$

Here  $\theta = \phi = 0$  by default.

The code for the  $U_3$  Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(1, 'q')
5 circuit = QuantumCircuit(qreg_q, name='U3 Gate')
6
7 circuit.u(pi/2, pi/2, pi/2, qreg_q[0]) #Apply the U3 gate
8
9 circuit.draw(output='mpl')#.savefig ('../images/u3-gate.png') #Draw the
circuit

```

The circuit symbol is as shown in figure 3.15.



Figure 3.15:  $U_3$  Gate

Note here the values  $\pi/2, \pi/2, \pi/2$  denotes the angles  $\theta, \phi, \lambda$  by which we rotate around the Bloch Sphere respectively.

### 3.3 Multi-Qubit Gates

Quantum Pauli gates and hadmard gate are Unitary and Hermitian, and all the other single qubit gate are Unitary, hence, reversible. In general, we can generalise any single qubit gate to multi-qubit gate. Note that since all the gates can be in general represented as Unitary Matrices, thus the number of qubits in input = number of qubits in output always. Multi-Qubit gates act on multiple qubits at once thus the matrices are of size  $2^n \times 2^n$  where  $n$  is the number of qubits.

#### 3.3.1 Two-qubit Gates

##### CNOT/CX Gate

It is a two qubit gate, also called as the Controlled-NOT gate. Here the 1st Qubit is controlled and the 2nd Qubit is target. (It is also possible in the other way that the 2nd Qubit is controlled and the 1st Qubit is target). This gate is used to create entanglement. The truth table is as in table 3.11.

Input	Output
$ 00\rangle$	$ 00\rangle$
$ 01\rangle$	$ 01\rangle$
$ 10\rangle$	$ 11\rangle$
$ 11\rangle$	$ 10\rangle$

Table 3.11: Truth Table for CNOT Gate

For the table in 3.11, the 1st Qubit is control and the 2nd Qubit is target. Thus, the CNOT gate flips the target qubit if the control qubit is  $|1\rangle$ . The code for the CNOT Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 # Add 2 qubits for CNOT gate demo
5 qreg_q = QuantumRegister(2, 'q')
6 circuit = QuantumCircuit(qreg_q, name='CNOT Gate')
7
8 circuit.cx(qreg_q[0], qreg_q[1]) #Apply the CNOT gate
9
10 circuit.draw(output='mpl')#.savefig ('../images/cnot-gate.png') #Draw
    the circuit

```

The circuit symbol is as shown in figure 3.16.

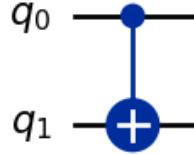


Figure 3.16: CNOT Gate

Here it can be seen that the 1st Qubit is control and the 2nd Qubit is target. In the figure 3.16, the dot denotes the control qubit and the cross denotes the target qubit.

In the operator Notation it is:

$$\begin{aligned} CX |00\rangle &= |00\rangle \\ CX |01\rangle &= |01\rangle \\ CX |10\rangle &= |11\rangle \\ CX |11\rangle &= |10\rangle \end{aligned}$$

It is represented by the matrix in the computational basis ( $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ ) as:

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

In Outer product representation, it can be written as:

$$CX = |00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 11| + |11\rangle\langle 10|$$

In Compact form we can write the CNOT gate as:

$$CX |xy\rangle = |x(x \oplus y)\rangle$$

where  $\oplus$  denotes the XOR operation. It is a non-linear operation. It is used in Quantum Error Correction Codes. This is a very important gate which is used in

Teleoprtation protocol, Superdense Coding, Quantum Error Correction Codes etc. Action of CX/CNOT on general two - qubit  $|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$  is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \delta \\ \gamma \end{bmatrix} = \alpha|00\rangle + \beta|01\rangle + \delta|10\rangle + \gamma|11\rangle$$

### CY Gate

It is a two qubit gate, also called as the Controlled-Y gate. Here the 1st Qubit is controlled and the 2nd Qubit is target (Pauli - Y Gate) (It is also possible in the other way that the 2nd Qubit is controlled and the 1st Qubit is target).

The truth table is as in table 3.12.

Input	Output
$ 00\rangle$	$ 00\rangle$
$ 01\rangle$	$ 01\rangle$
$ 10\rangle$	$i 10\rangle$
$ 11\rangle$	$-i 11\rangle$

Table 3.12: Truth Table for CY Gate

For the table in 3.12, the 1st Qubit is control and the 2nd Qubit is target. Thus, the CY gate flips the target qubit if the control qubit is  $|1\rangle$ . The code for the CY Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 # Add 2 qubits for CNOT gate demo
5 qreg_q = QuantumRegister(2, 'q')
6 circuit = QuantumCircuit(qreg_q, name='CY Gate')
7
8 circuit.cy(qreg_q[0], qreg_q[1]) #Apply the CY gate
9
10 circuit.draw(output='mpl')#.savefig ('../images/cy-gate.png') #Draw the
    circuit

```

The circuit symbol is as shown in figure 3.17.

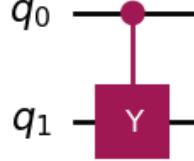


Figure 3.17: CY Gate

In this figure 3.17, the first qubit is control and the second qubit is target. If the first qubit is  $|1\rangle$ , then Y gate is applied on the second qubit. The Y Gate is a gate which rotates the qubit by  $\pi$  around the Y axis of Bloch Sphere. Thus, if the input is  $|0\rangle$ , then the output is  $|0\rangle$  and if the input is  $|1\rangle$ , then the output is  $-i|1\rangle$ .

In the operator Notation it is:

$$\begin{aligned} CY|00\rangle &= |00\rangle \\ CY|01\rangle &= |01\rangle \\ CY|10\rangle &= i|11\rangle \\ CY|11\rangle &= -i|10\rangle \end{aligned}$$

It is represented by the matrix in the computational basis ( $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ ) as:

$$CY = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{bmatrix}$$

In Outer product representation, it can be written as:

$$CY = |00\rangle\langle 00| + |01\rangle\langle 01| + i|11\rangle\langle 10| - i|10\rangle\langle 11|$$

In Compact form we can write the CY gate as:

$$CY|xy\rangle = (-1)^y i^x |x(x \oplus y)\rangle$$

Flipping amplitude of 1 state. Marking of this state can be utilised for marking elements in data base. It is useful in Grover's Algorithm.

Action of CY on general two - qubit  $|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$  is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\iota \\ 0 & 0 & \iota & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \iota\delta \\ -\iota\gamma \end{bmatrix} = \alpha|00\rangle + \beta|01\rangle - \iota\delta|10\rangle + \iota\gamma|11\rangle$$

### CZ/CPHASE Gate

It is a two qubit gate, also called as the Controlled-Z gate. Here the 1st Qubit is controlled and the 2nd Qubit is target (Pauli - Z Gate) (It is also possible in the other way that the 2nd Qubit is controlled and the 1st Qubit is target).

The truth table is as in table 3.13.

Input	Output
$ 00\rangle$	$ 00\rangle$
$ 01\rangle$	$ 01\rangle$
$ 10\rangle$	$ 10\rangle$
$ 11\rangle$	$- 11\rangle$

Table 3.13: Truth Table for CZ Gate

For the table in 3.13, the 1st Qubit is control and the 2nd Qubit is target. Thus, the CZ gate flips the target qubit if the control qubit is  $|1\rangle$ . The code for the CZ Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(2, 'q')
5 circuit = QuantumCircuit(qreg_q, name='CZ Gate')
6
7 circuit.cz(qreg_q[0], qreg_q[1]) #Apply the CZ gate
8
9 circuit.draw(output='mpl')#.savefig ('../images/CZ-gate.png') #Draw the
  circuit

```

The circuit symbol is as shown in figure 3.18.

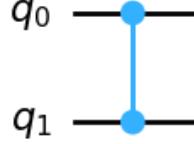


Figure 3.18: CZ Gate

In this figure 3.18, the first qubit is control and the second qubit is target. If the first qubit is  $|1\rangle$ , then Z gate is applied on the second qubit. The Z Gate is a gate which rotates the qubit by  $\pi$  around the Z axis of Bloch Sphere. Thus, if the input is  $|0\rangle$ , then the output is  $|0\rangle$  and if the input is  $|1\rangle$ , then the output is  $-|1\rangle$ .

In the operator Notation it is:

$$\begin{aligned} CZ|00\rangle &= |00\rangle \\ CZ|01\rangle &= |01\rangle \\ CZ|10\rangle &= |10\rangle \\ CZ|11\rangle &= -|11\rangle \end{aligned}$$

It is represented by the matrix in the computational basis ( $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ ) as:

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

In Outer product representation, it can be written as:

$$CZ = |00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 10| - |11\rangle\langle 11|$$

In Compact form we can write the CZ gate as:

$$CZ|xy\rangle = (-1)^{xy}|xy\rangle$$

Action of CZ on general two - qubit  $|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$  is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ -\delta \end{bmatrix} = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle - \delta|11\rangle$$

### CH/ Controlled Hadamard Gate

It is a two qubit gate, also called as the Controlled-Hadamard gate. Here the 1st Qubit is controlled and the 2nd Qubit is target. The truth table is as in table 3.14.

Input	Output
$ 00\rangle$	$ 00\rangle$
$ 01\rangle$	$ 01\rangle$
$ 10\rangle$	$\frac{1}{\sqrt{2}}( 10\rangle +  11\rangle)$
$ 11\rangle$	$\frac{1}{\sqrt{2}}( 10\rangle -  11\rangle)$

Table 3.14: Truth Table for CH Gate

For the table in 3.14, the 1st Qubit is control and the 2nd Qubit is target. Thus, the CH gate applies Hadamard gate on the target qubit if the control qubit is  $|1\rangle$ . The code for the CH Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 # Add 2 qubits for CNOT gate demo
5 qreg_q = QuantumRegister(2, 'q')
6 circuit = QuantumCircuit(qreg_q, name='CY Gate')
7
8 circuit.ch(qreg_q[0], qreg_q[1]) #Apply the CH gate
9
10 circuit.draw(output='mpl')#.savefig ('../images/ch-gate.png') #Draw the
    circuit

```

The circuit symbol is as shown in figure 3.19.

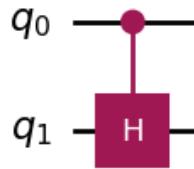


Figure 3.19: CH Gate

In this figure 3.19, the first qubit is control and the second qubit is target. If the first qubit is  $|1\rangle$ , then Hadamard gate is applied on the second qubit. The Hadamard Gate is a gate which rotates the qubit by  $\pi$  around the X axis of Bloch Sphere. Thus, if the input is  $|0\rangle$ , then the output is  $|0\rangle$  and if the input is  $|1\rangle$ , then the output is  $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ .

In the operator Notation it is:

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

It is represented by the matrix in the computational basis  $(|00\rangle, |01\rangle, |10\rangle, |11\rangle)$  as:

$$CH = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

In Outer product representation, it can be written as:

$$CH = |00\rangle\langle 00| + |01\rangle\langle 01| + \frac{1}{\sqrt{2}}|10\rangle\langle 10| + \frac{1}{\sqrt{2}}|10\rangle\langle 11|$$

In Compact form we can write the CH gate as:

$$CH|xy\rangle = \left(\frac{1}{\sqrt{2}}\right)^x (|x(x \oplus y)\rangle + (-1)^y |xy\rangle)$$

Action of CH on general two - qubit  $|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$  is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \frac{\gamma+\delta}{\sqrt{2}} \\ \frac{\gamma-\delta}{\sqrt{2}} \end{bmatrix} = \alpha|00\rangle + \beta|01\rangle + \frac{\gamma+\delta}{\sqrt{2}}|10\rangle + \frac{\gamma-\delta}{\sqrt{2}}|11\rangle$$

### Important Note

The CX/CNOT, CY, CZ and CH gates are Unitary as well as Hermitian. In general we can write any of the controlled gates where the first qubit acts as a control bit and the second qubit acts as a target bit in matrix form as show:

$$\begin{bmatrix} I & 0 \\ 0 & U \end{bmatrix}$$

where  $I$  is the Identity matrix of size  $2(2^{n-1} - 1) \times 2(2^{n-1} - 1)$  and  $U$  is the Unitary matrix of size  $2 \times 2$ . Now for forming any of the n-qubit control gates where there are  $n-1$  control bits and 1 target bit we replace  $U$  with the corresponding gate which we wish to form. For example, for 2-qubit Control Hadamard gate we replace  $U$  with the Hadamard gate matrix.

$$\begin{bmatrix} I & 0 \\ 0 & H \end{bmatrix}$$

Similarly, for any n-qubit control gate we can form the matrix by replacing  $U$  with the corresponding gate matrix. For example, for 3-qubit control Z gate we replace  $U$  with the Z gate matrix.

$$\begin{bmatrix} I & 0 \\ 0 & Z \end{bmatrix}$$

where  $I$  is the Identity matrix of size  $6 \times 6$  and  $Z$  is the Pauli-Z gate matrix of size  $2 \times 2$ . We will later on use this concept to create the CCNOT/Toffoli gate.

### SWAP Gate

It is a two Qubit Gate which swaps the states of the two qubits. The truth table is as in table 3.15.

Input	Output
$ 00\rangle$	$ 00\rangle$
$ 01\rangle$	$ 10\rangle$
$ 10\rangle$	$ 01\rangle$
$ 11\rangle$	$ 11\rangle$

Table 3.15: Truth Table for SWAP Gate

The code for the SWAP Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 # Add 2 qubits for CNOT gate demo
5 qreg_q = QuantumRegister(2, 'q')
6 circuit = QuantumCircuit(qreg_q, name='SWAP Gate')
7
8 circuit.swap(qreg_q[0], qreg_q[1]) #Apply the SWAP gate
9
10 circuit.draw(output='mpl')#.savefig ('../images/swap-gate.png') #Draw
    the circuit

```

The circuit symbol is as shown in figure 3.20.

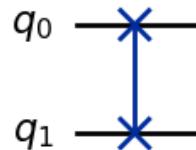


Figure 3.20: SWAP Gate

In the operator Notation it is:

$$\begin{aligned}
 SWAP |00\rangle &= |00\rangle \\
 SWAP |01\rangle &= |10\rangle \\
 SWAP |10\rangle &= |01\rangle \\
 SWAP |11\rangle &= |11\rangle
 \end{aligned}$$

It is represented by the matrix in the computational basis ( $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ ) as:

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

In Outer product representation, it can be written as:

$$SWAP = |00\rangle\langle 00| + |01\rangle\langle 10| + |10\rangle\langle 01| + |11\rangle\langle 11|$$

Action of SWAP on general two - qubit  $|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$  is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} = \begin{bmatrix} \alpha \\ \gamma \\ \beta \\ \delta \end{bmatrix} = \alpha|00\rangle + \gamma|01\rangle + \beta|10\rangle + \delta|11\rangle$$

SWAP Gate can also be formed using CNOT gates as follows:

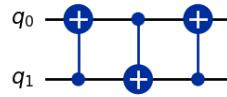


Figure 3.21: SWAP Gate using CNOT Gates

Or using CNOT gates as follows:

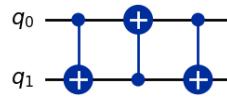


Figure 3.22: SWAP Gate using CNOT Gates

### 3.3.2 Three-qubit Gates

#### CCNOT/Toffoli Gate

It is a three qubit gate, also called as the CCNOT gate. Here the 1st and 2nd Qubits are controlled and the 3rd Qubit is target. The truth table is as in table 3.16.

Input	Output
$ 000\rangle$	$ 000\rangle$
$ 001\rangle$	$ 001\rangle$
$ 010\rangle$	$ 010\rangle$
$ 011\rangle$	$ 011\rangle$
$ 100\rangle$	$ 100\rangle$
$ 101\rangle$	$ 101\rangle$
$ 110\rangle$	$ 111\rangle$
$ 111\rangle$	$ 110\rangle$

Table 3.16: Truth Table for Toffoli Gate

For the table in 3.16, the 1st and 2nd Qubits are control and the 3rd Qubit is target. Thus, the Toffoli gate flips the target qubit if the control qubits are  $|1\rangle$ . The code for the Toffoli Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(3, 'q')
5 circuit = QuantumCircuit(qreg_q)
6
7 circuit.ccx(qreg_q[0], qreg_q[1], qreg_q[2])
8
9 circuit.draw(output='mpl')#.savefig ('../images/ccx-gate.png')

```

The circuit symbol is as shown in figure 3.23.

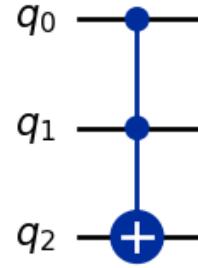


Figure 3.23: Toffoli Gate

In this figure 3.23, the first and second qubits are control and the third qubit is target. If the first and second qubits are  $|1\rangle$ , then the third qubit is flipped. The Toffoli Gate is a gate which flips the target qubit if the control qubits are  $|1\rangle$ .

In the operator Notation it is:

$$\begin{aligned}
 CCX |000\rangle &= |000\rangle \\
 CCX |001\rangle &= |001\rangle \\
 CCX |010\rangle &= |010\rangle \\
 CCX |011\rangle &= |011\rangle \\
 CCX |100\rangle &= |100\rangle \\
 CCX |101\rangle &= |101\rangle \\
 CCX |110\rangle &= |111\rangle \\
 CCX |111\rangle &= |110\rangle
 \end{aligned}$$

It is represented by the matrix in the computational basis ( $|000\rangle, |001\rangle, \dots, |110\rangle, |111\rangle$ )

as:

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

In Outer product representation, it can be written as:

$$\begin{aligned} CCX = & |000\rangle\langle 000| + |001\rangle\langle 001| + |010\rangle\langle 010| + |011\rangle\langle 011| \\ & + |100\rangle\langle 100| + |101\rangle\langle 101| + |110\rangle\langle 110| + |111\rangle\langle 111| \end{aligned}$$

Action of CCX on general three - qubit  $|\psi\rangle = \alpha|000\rangle + \beta|001\rangle + \gamma|010\rangle + \delta|011\rangle + \epsilon|100\rangle + \zeta|101\rangle + \eta|110\rangle + \theta|111\rangle$  is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \\ \epsilon \\ \zeta \\ \eta \\ \theta \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \\ \epsilon \\ \zeta \\ \eta \\ \theta \end{bmatrix} = \alpha|000\rangle + \beta|001\rangle + \dots + \zeta|101\rangle + \theta|110\rangle + \eta|111\rangle$$

### CSWAP/Fredkin Gate

It is a three qubit gate, also called as the CSWAP gate. Here the 1st Qubit is control and the 2nd and 3rd Qubits are target. The truth table is as in table 3.17.

Input	Output
$ 000\rangle$	$ 000\rangle$
$ 001\rangle$	$ 001\rangle$
$ 010\rangle$	$ 010\rangle$
$ 011\rangle$	$ 011\rangle$
$ 100\rangle$	$ 100\rangle$
$ 101\rangle$	$ 110\rangle$
$ 110\rangle$	$ 101\rangle$
$ 111\rangle$	$ 111\rangle$

Table 3.17: Truth Table for Fredkin Gate

For the table in 3.17, the 1st Qubit is control and the 2nd and 3rd Qubits are target. Thus, the Fredkin gate swaps the states of the 2nd and 3rd qubits if the control qubit is  $|1\rangle$ . The code for the Fredkin Gate is as follows:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(3, 'q')
5 circuit = QuantumCircuit(qreg_q)
6
7 circuit.cswap(qreg_q[0], qreg_q[2], qreg_q[1])
8
9 circuit.draw(output='mpl')#.savefig ('../images/cswap-gate.png')

```

The circuit symbol is as shown in figure 3.24.

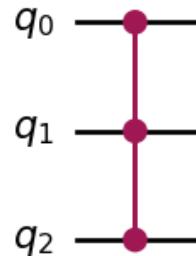


Figure 3.24: Fredkin Gate

In this figure 3.24, the first qubit is control and the second and third qubits are target. If the first qubit is  $|1\rangle$ , then the second and third qubits are swapped. The Fredkin Gate is a gate which swaps the states of the 2nd and 3rd qubits if the control qubit is  $|1\rangle$ .

In the operator Notation it is:

$$\begin{aligned}
 CSWAP |000\rangle &= |000\rangle \\
 CSWAP |001\rangle &= |001\rangle \\
 CSWAP |010\rangle &= |010\rangle \\
 CSWAP |011\rangle &= |011\rangle \\
 CSWAP |100\rangle &= |100\rangle \\
 CSWAP |101\rangle &= |110\rangle \\
 CSWAP |110\rangle &= |101\rangle \\
 CSWAP |111\rangle &= |111\rangle
 \end{aligned}$$

It is represented by the matrix in the computational basis ( $|000\rangle, |001\rangle, \dots, |110\rangle, |111\rangle$ )

as:

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

In Outer product representation, it can be written as:

$$\begin{aligned} CSWAP = & |000\rangle\langle 000| + |001\rangle\langle 001| + |010\rangle\langle 010| + |011\rangle\langle 011| \\ & + |100\rangle\langle 100| + |101\rangle\langle 110| + |110\rangle\langle 101| + |111\rangle\langle 111| \end{aligned}$$

Action of CSWAP on general three - qubit  $|\psi\rangle = \alpha|000\rangle + \beta|001\rangle + \gamma|010\rangle + \delta|011\rangle + \epsilon|100\rangle + \zeta|101\rangle + \eta|110\rangle + \theta|111\rangle$  is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \\ \epsilon \\ \zeta \\ \eta \\ \theta \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \\ \epsilon \\ \zeta \\ \eta \\ \theta \end{bmatrix} = \alpha|000\rangle + \beta|001\rangle + \dots + \zeta|101\rangle + \theta|110\rangle + \eta|111\rangle$$

We can construct Fredkin gate using CNOT and Toffoli gate as shown in the figure 3.25.

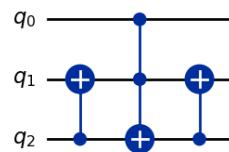


Figure 3.25: Fredkin Gate using CNOT and Toffoli Gates

### Important Note

Applying a gate on a superposition state is the same as applying the gate on each of the basis states and then superposing the results. This is because, the gate is a linear operator and the superposition is a linear combination of the basis states. Thus, the gate can be applied on each of the basis states and then the results can be superposed. For example, Consider a superposition state  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ . If we apply a gate  $U$  on this state, then the output will be  $U|\psi\rangle = U(\alpha|0\rangle + \beta|1\rangle) = \alpha U|0\rangle + \beta U|1\rangle$ . Thus, the gate can be applied on each of the basis states and then the results can be superposed. This, can also be thought that since  $U$  is a  $2 \times 2$  matrix and  $|0\rangle$  and  $|1\rangle$  are  $2 \times 1$  column vectors, multiplying  $U(\alpha|0\rangle + |1\rangle)$  is same as  $\alpha U|0\rangle + \beta U|1\rangle$ . Since Matrix multiplication is distributive.

For example, say CNOT gate needs to be applied on a superposition of two bits  $|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ . Then the output will be  $CNOT|\psi\rangle = CNOT\left(\frac{|00\rangle + |11\rangle}{\sqrt{2}}\right) = \frac{CNOT|00\rangle + CNOT|11\rangle}{\sqrt{2}}$ . which will thus be  $\frac{|00\rangle + |10\rangle}{\sqrt{2}}$ . In the matrix form,

$$\begin{aligned} CNOT|\psi\rangle &= CNOT\left(\frac{|00\rangle + |11\rangle}{\sqrt{2}}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \frac{1}{\sqrt{2}} \left( \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right) \\ &= \frac{1}{\sqrt{2}} \left( \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right) \\ &= (CNOT\frac{1}{\sqrt{2}}|00\rangle + CNOT\frac{1}{\sqrt{2}}|11\rangle) = \frac{1}{\sqrt{2}}CNOT|00\rangle + \frac{1}{\sqrt{2}}CNOT|11\rangle \\ &= \frac{1}{\sqrt{2}} \left( \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) \end{aligned}$$

Thus, applying a gate on a superposition is the same as applying the gate on the basis states of the superposition. Thus, the gate can be applied on each of the basis states and then the results can be superposed in the same linear combination as that in which the state  $|\psi\rangle$  was superposed.

## 3.4 Quantum Circuits

The allowed operations on (pure) quantum states are those that map normalized states to normalized states, namely unitary operators  $U$ , satisfying  $UU^\dagger = U^\dagger U = I$ . To have a sensible notion of efficient computation, we require that the unitary operators appearing in a quantum computation are realized by quantum circuits. We are given a set of gates, each of which acts on one or two qubits at a time (meaning that it is a tensor product of one or two qubit operator with the identity operator on the remaining qubits). A quantum computation begins in the  $|0\rangle$  state, applies a sequence of one and two qubit gates chosen from the set of allowed gates, and finally reports an outcome obtained by measuring in the computational basis.

A quantum circuit (also called quantum network or quantum gate array) generalizes the idea of classical circuit families, replacing the AND, OR, and NOT gates by elementary quantum gates. A quantum gate is a unitary transformation on a small (usually 1, 2, or 3) number of qubits. We saw a number of examples already: the bitflip gate  $X$ , the phaseflip gate  $Z$ , the Hadamard gate  $H$ . The main 2-qubit gate we have seen is the controlled-NOT (CNOT) gate. Adding another control qubit, we get the 3-qubit Toffoli gate, also called controlled-controlled-not (CCNOT) gate. This negates the third bit of its input if both of the first two bits are 1. The Toffoli gate is important because it is complete for classical reversible computation: any classical computation can be implemented by a circuit of Toffoli gates. This is easy to see: using auxiliary wires with fixed values, Toffoli can implement AND (fix the 3rd ingoing wire to 0) and NOT (fix the 1st and 2nd ingoing wire to 1). It is known that AND and NOT-gates together sufficient to implement any classical Boolean circuit, so if we can apply (or simulate) Toffoli gates, we can implement any classical computation in a reversible manner.

Mathematically, such elementary quantum gates can be composed into bigger unitary operations by taking tensor products (if gates are applied in parallel to different parts of the register), and ordinary matrix products (if gates are applied sequentially). We have already seen a simple example of such a circuit of elementary gates in the previous chapter, namely to implement teleportation.

As in the classical case, a quantum circuit is a finite directed acyclic graph of input nodes, gates, and output node. There are  $n$  nodes that contain the input (as classical bits); in addition we may have some more input nodes that are initially  $|0\rangle$  ("workspace"). The internal nodes of the quantum circuit are quantum gates that each operate on at most two or three qubits of the state. The gates in the circuit transform the initial state vector into a final state, which will generally be a superposition. We measure some or all qubits of this final state in the computational

basis in order to (probabilistically) obtain a classical output to the algorithm. We can think of the measurement of one qubit in the computational basis as one special type of gate. We may assume without much loss of generality that such measurements only happen at the very end of the circuit.

What about the more general kinds of measurements? If we want to apply such a measurement in the circuit model, we will have to implement it using a circuit of elementary gates followed by a measurement in the computational basis. For example, suppose projectors  $P_0$  and  $P_1$  both have rank  $2^n/2$ . Then there exists a unitary  $U$  that maps an  $n$ -qubit state  $|\phi\rangle$  to a state whose first qubit is  $|1\rangle$  whenever  $P_1|\psi\rangle = |\psi\rangle$ . We can now implement the projective measurement by first applying a circuit that implements  $U$ , and then measuring (in the computational basis) the first qubit of the resulting state. The minimal-size circuit to implement  $U$  could be very large (i.e., expensive) if the projective measurement is complicated, but that is how it should be.

To draw quantum circuits, the convention is to let time progress from left to right: we start with the initial state on the left. Each qubit is pictured as a horizontal wire, and the circuit prescribes which gates are to be applied to which wires. Single-qubit gates like X and H just act on one wire, while multi-qubit gates such as the CNOT act on multiple wires simultaneously. When one qubit "controls" the application of a gate to another qubit, then the controlling wires is drawn with a dot linked vertically to the gate that is applied to the target qubit. This happens for instance with the CNOT, where the applied single-qubit gate is X, usually drawn as  $\oplus$  in a circuit picture (similarly, the Toffoli gate is drawn in a circuit with a dot on the two control wire and an  $\oplus$  on the target wires).

Note that if we have a circuit for unitary  $U$ , it is very easy to find a circuit for the inverse  $U^{-1}$  with the same complexity: just reverse the order of the gates, and take the inverse of each gate. For example, if  $U = U_1 U_2 U_3$ , then  $U^{-1} = U_3^{-1} U_2^{-1} U_1^{-1}$ .

**Example 3.4.1.** Consider the quantum circuit as shown in figure 3.26.

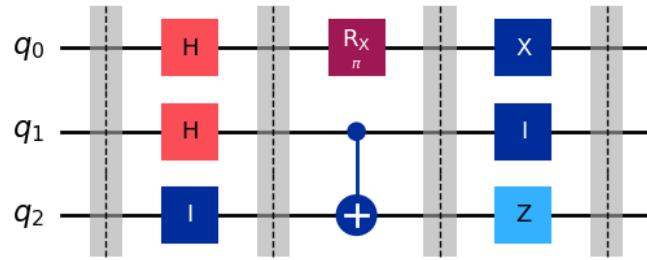


Figure 3.26: Quantum Circuit

1. Make Reverse circuit of the given circuit.
  2. Write the output of the circuit for the input  $|010\rangle$ .
  3. Write complete circuit as Unitary Matrix.
  4. Write reverse circuit as Unitary Matrix.
  5. Compute output using Unitary matrix of circuit.
1. Note that except for the rotation gate all the other gates are their own inverses.  
Thus, the reverse circuit is as shown in figure 3.27.

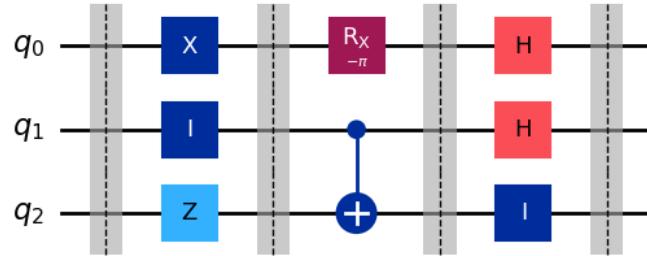


Figure 3.27: Reverse Quantum Circuit

If we place the two circuits side by side, we will get the input back. Thus it will act like an Identity matrix.

2. As shown in the figure 3.26 the circuit has been divided into parts for easy computation. At the first barrier, the input is  $|010\rangle$ . After applying the gates, at the second stage the output will be:

$$\begin{aligned}(H \otimes H \otimes I)(|010\rangle) &= H|0\rangle \otimes H|1\rangle \otimes I|0\rangle \\&= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes |0\rangle \\&= \frac{1}{2}(|000\rangle - |010\rangle + |100\rangle - |110\rangle)\end{aligned}$$

After this, the third stage requires the application of the RX gate on the first qubit and CNOT gate with 2nd qubit as control and 3rd qubit as target. The output after this stage will be:

$$\begin{aligned}(RX \otimes CX)\left(\frac{1}{2}(|000\rangle - |010\rangle + |100\rangle - |110\rangle)\right) &= \frac{1}{2}(RX|0\rangle \otimes CX|00\rangle - RX|0\rangle \otimes CX|10\rangle \\&\quad + RX|1\rangle \otimes CX|00\rangle - RX|1\rangle \otimes CX|10\rangle) \\&= \frac{|100\rangle - |111\rangle - |000\rangle + |011\rangle}{2}\end{aligned}$$

Here the action of RXx is  $RX|0\rangle = |1\rangle$ ,  $RX|1\rangle = -|0\rangle$ . Now we apply the last stage which involves applying X gate on the first qubit and Z gate on the 3rd qubits.

$$\begin{aligned}(X \otimes I \otimes Z)\frac{|100\rangle - |111\rangle - |000\rangle + |011\rangle}{2} &= \frac{1}{2}(X|1\rangle \otimes I|0\rangle \otimes Z|0\rangle - X|1\rangle \otimes \\&\quad I|1\rangle \otimes Z|1\rangle - X|0\rangle \otimes I|0\rangle \otimes Z|0\rangle + X|0\rangle \otimes I|1\rangle \\&\quad \otimes Z|1\rangle) \\&= \frac{1}{2}(|000\rangle + |011\rangle - |100\rangle - |111\rangle)\end{aligned}$$

Thus, the state of the qubits after passing through the circuit. In the matrix form, it can be written as:

$$\frac{1}{2} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \\ -1 \end{bmatrix}$$

3. In order to find the Unitary matrix for the complexe circuit, we are required to find the unitary matrices corresponding to the individual stages and multiply them to get the final Unitary Matrix. For the first stage, we have two Hadamard gates acting on first and second qubit and an Identity gate. Thus, we are required to find,

$$\begin{aligned} S_1 &= H \otimes H \otimes I \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

Thus, doing this the overall matrix will be:

$$S_1 = \frac{1}{2} \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \end{bmatrix}$$

For stage 2, we are required to find,

$$\begin{aligned} S_2 &= RX \otimes CX \\ &= \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{aligned}$$

In the matrix form, we get:

$$S_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

For stage 3, we are required to find,

$$\begin{aligned}
 S_3 &= X \otimes I \otimes Z \\
 &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\
 &= \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}
 \end{aligned}$$

Finally, we are required to calculate the product of the three matrices to get the overall matrix.

$$\begin{aligned}
 U &= S_3 S_2 S_1 \\
 &= \frac{1}{2} \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 \\ -1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \end{bmatrix}
 \end{aligned}$$

- The Unitary matrix for the reverse circuit will be the Transpose conjugate of the given circuit i.e. the transpose conjugate of the matrix found in the previous part.

$$U^\dagger = \frac{1}{2} \begin{bmatrix} 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & 1 & -1 & 0 & 0 & -1 \\ 0 & -1 & -1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \\ 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & -1 & -1 & 0 & 0 & -1 & -1 & 0 \end{bmatrix}$$

Thus, the matrix for the reverse circuit.

5. The output of the circuit can be found by multiplying the input with the Unitary matrix of the circuit.

$$U \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \\ -1 \end{bmatrix}$$

which can be verified as the same output from the previous part.

# Chapter 4

## Entanglement

Given a state of multiple entangled qubit, one cannot express individual qubit separately. For example, consider the state  $|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$ , cannot be expressed as some  $|\phi\rangle \otimes |\chi\rangle$ ,  $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$  and  $|\chi\rangle = \gamma|0\rangle + \delta|1\rangle$  i.e. no value of  $\alpha, \beta, \gamma, \delta$  exists. The states are called entangled.

On the contrary some states can be separated into individual qubit states, such states are called separable states. For example, consider the state  $|\psi\rangle = \frac{|00\rangle + |01\rangle}{\sqrt{2}}$ , can be expressed as  $|0\rangle \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}}$ , here  $\alpha = 1, \beta = 0, \gamma = \frac{1}{\sqrt{2}}, \delta = \frac{1}{\sqrt{2}}$ .

Given an Entangled state of multiple qubit, measuring any qubit individually reveals all other qubits. For example, consider  $|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$ , thus suppose we measure only one qubit, i.e. say we measure the first qubit. Then the probability of the first qubit being  $|0\rangle$  is  $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$  and the probability of the first qubit being  $|1\rangle$  is  $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$ . Now suppose we find that the first qubit is  $|0\rangle$ , then the state of the quantum system after measurement will be  $|00\rangle$ , thus the state of the second qubit will be  $|0\rangle$ . Similarly, if we find that the first qubit is  $|1\rangle$ , then the state of the quantum system after measurement will be  $|11\rangle$ , thus the state of the second qubit will be  $|1\rangle$ . This, we can see that measuring one qubit reveals the state of the other qubit. This is called entanglement.

But, in case of separable qubits measuring one qubit will not reveal the state of the second qubit. For example, consider the state  $|\psi\rangle = \frac{|00\rangle + |01\rangle}{\sqrt{2}}$ , if we measure the first qubit, then the probability of the first qubit being  $|0\rangle$  is  $|\frac{1}{\sqrt{2}}|^2 + |\frac{1}{\sqrt{2}}|^2 = 1$  and the probability of the first qubit being  $|1\rangle$  is 0. Thus, if we find that the first qubit is  $|0\rangle$ , then the state of the quantum system after measurement will be  $\frac{|00\rangle + |01\rangle}{\sqrt{2}}$ , thus the state of the second qubit will be  $|0\rangle$  with probability  $\frac{1}{2}$  and will be in state  $|1\rangle$  with probability  $\frac{1}{2}$ . Thus, the state of the second qubit cannot be revealed. The

measurement on the first qubit does not reveal any information of the state of the second qubit.

## 4.1 Bell States/EPR Pairs

We have four Bell states, which are also called as EPR states. These are:

$$\begin{aligned} |\Phi^+\rangle &= |\beta_{00}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \\ |\Phi^-\rangle &= |\beta_{10}\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}} \\ |\Psi^+\rangle &= |\beta_{01}\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} \\ |\Psi^-\rangle &= |\beta_{11}\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \end{aligned}$$

The general formula for Bell states is:

$$|\beta_{xy}\rangle = \frac{|0y\rangle + (-1)^x |1\bar{y}\rangle}{\sqrt{2}}$$

where  $\bar{y}$  is the negation of  $y$ . Thus, we can find any Bell state using the above formula for any given input  $x \in \{0, 1\}$  and  $y \in \{0, 1\}$ . We can create these bell states using quantum circuits as shown in figure 4.1.

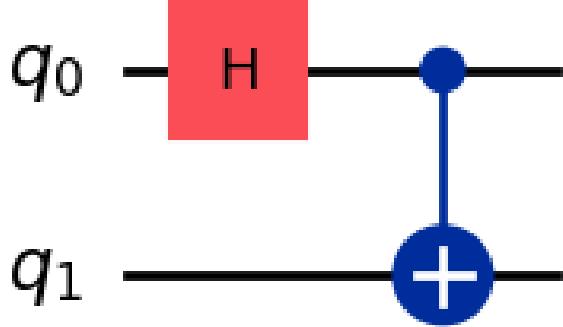


Figure 4.1: Bell States

For example, suppose we wish to apply the circuit on state  $|01\rangle$ . Then the state after applying the operation  $H \otimes I$  will be  $\frac{|0\rangle+|1\rangle}{\sqrt{2}} \otimes |1\rangle = \frac{|01\rangle+|11\rangle}{\sqrt{2}}$ . Similarly, the state after applying the operation  $CX$  will be  $\frac{|01\rangle+|10\rangle}{\sqrt{2}} = \beta_{01}$ .

**Example 4.1.1.** Given the state  $|\psi\rangle = \frac{|01\rangle-|10\rangle}{\sqrt{2}}$ , find whether the state is entangled or not. We prove this by contradiction. Assume that  $|\psi\rangle$  can be written separately as  $\alpha|0\rangle + \beta|1\rangle \otimes \gamma|0\rangle + \delta|1\rangle$ . Thus, we get,

$$\begin{aligned} \frac{|01\rangle - |10\rangle}{\sqrt{2}} &= \alpha|0\rangle + \beta|1\rangle \otimes \gamma|0\rangle + \delta|1\rangle \\ &= \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle \end{aligned}$$

Comparing the equations we have,

$$\begin{aligned}\alpha\gamma &= 0 \\ \alpha\delta &= \frac{1}{\sqrt{2}} \\ \beta\gamma &= -\frac{1}{\sqrt{2}} \\ \beta\delta &= 0\end{aligned}$$

No solution exists for the above equations, thus the given state is entangled as it cannot be written as a tensor product of two states.

Entanglement has applications in Teleportation and Superdense Coding which will be discussed next. To get back the original input state from the Bell states we can use the following Circuit.

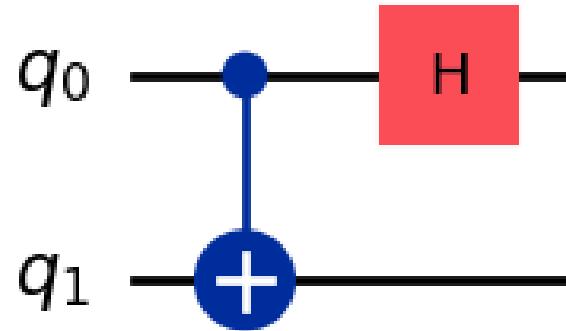


Figure 4.2: Reverse Bell States

Here, we have used a reverse CNOT gate i.e. inverse of CNOT gate which is the CNOT gate since it's a self-inverse gate (Hermitian and Unitary). Then we use the

reverse Hadamard gate which is Hadamard Gate on the first qubit and the circuit will give back the original input state. (Hadmand gate is also self-inverse).

## 4.2 No Cloning Theorem

One of the counter-intuitive consequences of the laws of Quantum Mechanics is that it is impossible to create an exact copy of an arbitrary unknown quantum state. This is called the No Cloning Theorem. More formally,

**Theorem 4.2.1.** *Given an arbitrary unknown quantum state  $|\psi\rangle$ , it is impossible to create an exact copy of the state  $|\psi\rangle$  i.e. create the state  $|\psi\rangle|\psi\rangle$ . More precisely, it is impossible to start with two qubits in the state  $|\phi\rangle\otimes|0\rangle$  and transform them to the state  $|\phi\rangle\otimes|\phi\rangle$ .*

*Proof.* From the Postulate 2: Evolution Postulate given in section 2.2, from Axiom of Quantum Mechanics, for this to be possible there must be a Unitary transformation  $U$  such that  $U|\phi\rangle\otimes|0\rangle = |\phi\rangle\otimes|\phi\rangle$ . Thus, in order to prove that it is impossible to do so, it suffices to prove that no such unitary transformation exists. We start with proving that no such unitary transformation exists that can achieve this simultaneously for two states  $|\phi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$  and  $|\psi\rangle = \beta_0|0\rangle + \beta_1|1\rangle$ .

Recall that a unitary transformation is a rotation of the Hilbert Space, and therefore necessarily preserves inner products. To be more precise, consider two quantum states:

$$|\phi\rangle\otimes|0\rangle = \alpha_0|00\rangle + \alpha_1|10\rangle$$

and

$$|\psi\rangle\otimes|0\rangle = \beta_0|00\rangle + \beta_1|10\rangle$$

Then, the inner product between them is given by

$$\langle\phi|0|\psi\rangle = \alpha_0^*\beta_0 + \alpha_1^*\beta_1 = \langle\phi|\psi\rangle$$

Now consider the quantum states,

$$\begin{aligned} |\phi\rangle\otimes|\phi\rangle &= (\alpha_0|0\rangle + \alpha_1|1\rangle)\otimes(\alpha_0|0\rangle + \alpha_1|1\rangle) \\ &= \alpha_0^2|00\rangle + \alpha_0\alpha_1|01\rangle + \alpha_1\alpha_0|10\rangle + \alpha_1^2|11\rangle \end{aligned}$$

and

$$\begin{aligned} |\psi\rangle\otimes|\psi\rangle &= (\beta_0|0\rangle + \beta_1|1\rangle)\otimes(\beta_0|0\rangle + \beta_1|1\rangle) \\ &= \beta_0^2|00\rangle + \beta_0\beta_1|01\rangle + \beta_1\beta_0|10\rangle + \beta_1^2|11\rangle \end{aligned}$$

Their inner product is

$$\begin{aligned}\langle\phi\phi|\psi\psi\rangle &= (\langle 00| \alpha_0^{*2} + \langle 01| \alpha_0^* \alpha_1^* + \langle 10| \alpha_1^* \alpha_0^* + \langle 11| \alpha_1^{*2})(\beta_0^2 |00\rangle + \beta_0 \beta_1 |01\rangle + \beta_1 \beta_0 |10\rangle + \beta_1^2 |11\rangle) \\ &= \alpha_0^{*2} \beta_0^2 + \alpha_0^* \alpha_1^* \beta_0 \beta_1 + \alpha_1^* \alpha_0 \beta_1 \beta_0 + \beta_0^2 \beta_1^2 \\ &= (\alpha_0^* \beta_0 + \alpha_1^* \beta_1)^2\end{aligned}$$

Now let us assume that we have a Unitary operator  $U$  and the two arbitrary quantum states  $|\phi\rangle$  and  $|\psi\rangle$  such that the action of  $U$  is defined as follows:

$$\begin{aligned}|\phi\rangle \otimes |0\rangle &\xrightarrow{U} |\phi\rangle \otimes |\phi\rangle \\ |\psi\rangle \otimes |0\rangle &\xrightarrow{U} |\psi\rangle \otimes |\psi\rangle\end{aligned}$$

Thus, clearly operator  $U$  corresponds to an operator which performs copying arbitrary quantum states. Since we are performing the same unitary transformation on both the states  $|\phi\rangle$  and  $|\psi\rangle$ , thus the inner product before and after transformation must be the same, as a Unitary transformation corresponds to rotations in the complex Hilbert space. Their inner product before transformation is

$$\langle\phi 0|\psi 0\rangle = \alpha_0^* \beta_0 + \alpha_1^* \beta_1$$

Their inner product after the transformation will be:

$$\langle\phi\phi|\psi\psi\rangle = (\alpha_0^* \beta_0 + \alpha_1^* \beta_1)^2 = \langle\phi 0|\psi 0\rangle^2 = \langle\phi|\psi\rangle^2$$

Thus, before transformation we had the inner product as  $\langle\phi|\psi\rangle$  and after the transformation we have the inner product as  $\langle\phi|\psi\rangle^2$ . Thus, in general,  $\langle\phi|\psi\rangle \neq \langle\phi|\psi\rangle^2$  (It is equal only in the case when  $\langle\phi|\psi\rangle = 0$  or  $1$  i.e. both the quantum states are orthonormal). Since, this is not true for any two arbitrary quantum states, thus our assumption that such a Unitary transformation  $U$  exists is wrong. Thus, by contradiction, we have proved that such a Unitary transformation  $U$  does not exist, hence, it is impossible to copy an arbitrary unknown quantum state  $|\phi\rangle$  or in other words, to be more precise, no such unitary transformation  $U$  exists which can perform the following transformation of  $|\phi\rangle \otimes |0\rangle \rightarrow |\phi\rangle \otimes |\phi\rangle$ .  $\square$

## 4.3 Applications of Entanglement

### 4.3.1 Quantum Teleportation

Alice and Bob are two friends. They get together and create a Bell state  $\beta_{00}$  and then distribute the quantum system. One qubit stays with Alice (say  $|\beta_A\rangle$ ) and the

other qubit goes to Bob (say  $|\beta_B\rangle$ ). Now Bob moves 100,000 light years away. Now, say Alice wishes to send Bob an Arbitrary state  $|\psi\rangle$  to Bob.

The teleportation protocol is as follows:

1. Alice and Bob create a Bell state  $\beta_{00}$ . It is as shown in the figure 4.3 which has both the inputs  $q_0 = |0\rangle$  and  $q_1 = |0\rangle$ .

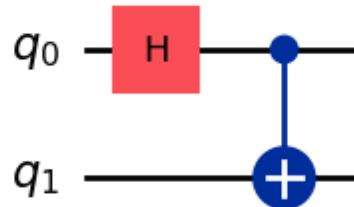


Figure 4.3: Creating Bell States

The state of the circuit after applying this state is the Bell state which is given as  $\beta_{00} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$ . Now, say the outcome of the first bit be  $|\beta_A\rangle$  which is the state of the first qubit goes to Alice and the outcome of the second bit be  $|\beta_B\rangle$  which is the state of the second qubit which goes to Bob.

2. Alice applies a CNOT gate with the input state  $|\psi\rangle$  as the control qubit and the state  $|\beta_A\rangle$  as the target qubit. Then Alice applies a Hadamard gate on the input state  $|\psi\rangle$ . This is as shown in the figure 4.4 with  $q_0 = |\psi\rangle$  and  $q_1 = |\beta_A\rangle$ .

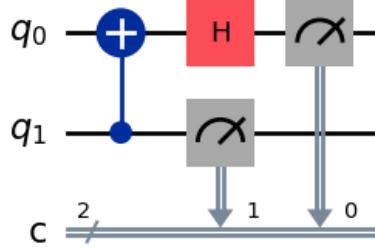


Figure 4.4: Alice's Operations

Thus the state here, is  $|\psi\rangle |\beta_{00}\rangle$ . Let the state  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ . Thus,

$$|\psi\rangle |\beta_{00}\rangle = (\alpha|0\rangle + \beta|1\rangle) \otimes \frac{|00\rangle + |11\rangle}{\sqrt{2}} = \frac{\alpha|000\rangle + \alpha|011\rangle + \beta|100\rangle + \beta|111\rangle}{\sqrt{2}}$$

Now applying CNOT with first bit as control bit and the second bit as the target bit we get,

$$\frac{\alpha|000\rangle + \alpha|011\rangle + \beta|110\rangle + \beta|101\rangle}{\sqrt{2}}$$

Now, applying Hadamard gate on the first bit we get,

$$\frac{\alpha|000\rangle + \alpha|100\rangle + \alpha|011\rangle + \alpha|111\rangle + \beta|010\rangle - \beta|110\rangle + \beta|001\rangle - \beta|101\rangle}{2}$$

We further simplify this by considering only the first two qubits, we get,

$$\frac{1}{2} [|00\rangle (\alpha|0\rangle + \beta|1\rangle) + |01\rangle (\alpha|1\rangle + \beta|0\rangle) + |10\rangle (\alpha|0\rangle - \beta|1\rangle) + |11\rangle (\alpha|1\rangle - \beta|0\rangle)]$$

Now Alice does measurement of the first two qubits. Remember that the qubit  $|\psi\rangle$  is already normalized i.e.  $|\alpha|^2 + |\beta|^2 = 1$ . The probability of the first two qubits being  $|00\rangle$  is  $|\frac{\alpha}{2}|^2 + |\frac{\beta}{2}|^2 = \frac{|\alpha|^2 + |\beta|^2}{4} = \frac{1}{4}$ , similarly, the probability of the first two qubits being  $|10\rangle$  is  $\frac{1}{4}$ , the probability of the first two qubits being  $|01\rangle$  is  $\frac{1}{4}$  and the probability of the first two qubits being  $|11\rangle$  is  $\frac{1}{4}$ .

Suppose upon measuring the two qubits by Alice they collapsed to  $|00\rangle$ , then the state of the system will be:

$$\alpha|000\rangle + \beta|001\rangle = |00\rangle \otimes (\alpha|0\rangle + \beta|1\rangle)$$

Suppose upon measuring the two qubits by Alice they collapsed to  $|10\rangle$ , then the state of the system will be:

$$\alpha|100\rangle - \beta|101\rangle = |10\rangle \otimes (\alpha|0\rangle - \beta|1\rangle)$$

Suppose upon measuring the two qubits by Alice they collapsed to  $|01\rangle$ , then the state of the system will be:

$$\alpha|011\rangle + \beta|010\rangle = |01\rangle \otimes (\alpha|1\rangle + \beta|0\rangle)$$

Suppose upon measuring the two qubits by Alice they collapsed to  $|11\rangle$ , then the state of the system will be:

$$\alpha|111\rangle - \beta|110\rangle = |11\rangle \otimes (\alpha|1\rangle - \beta|0\rangle)$$

All of the above are possible with a probability of  $\frac{1}{4}$ .

3. Now Alice sends the result of the measurement to Bob. Bob applies the necessary operations to get the original state  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ .

- Suppose Alice measured the first two qubits to be  $|00\rangle$ , then Bob applies no operation.

$$\alpha|000\rangle + \beta|001\rangle = |00\rangle \otimes (\alpha|0\rangle + \beta|1\rangle)$$

- Suppose Alice measured the first two qubits to be  $|10\rangle$ , then Bob applies the Z gate on his qubit to get the original state.

$$Z(\alpha|100\rangle - \beta|101\rangle) = \alpha|100\rangle + \beta|101\rangle = |10\rangle \otimes (\alpha|0\rangle + \beta|1\rangle)$$

- Suppose Alice measured the first two qubits to be  $|01\rangle$ , then Bob applies the X gate on his qubit to get the original state.

$$X(\alpha|011\rangle + \beta|010\rangle) = \alpha|010\rangle + \beta|011\rangle = |01\rangle \otimes (\alpha|0\rangle + \beta|1\rangle)$$

- Suppose Alice measured the first two qubits to be  $|11\rangle$ , then Bob applies the X and Z gate on his qubit to get the original state.

$$Z(X(\alpha|111\rangle - \beta|110\rangle)) = \alpha|110\rangle + \beta|111\rangle = |11\rangle \otimes (\alpha|0\rangle + \beta|1\rangle)$$

Thus, we have teleported the unknown state  $|\psi\rangle$  from Alice to Bob. Note that during this protocol, the Bell state of the qubits is destroyed upon measurement by Alice and thus, cannot be done again.

The final Teleportation circuit is as shown in the figure 4.5.

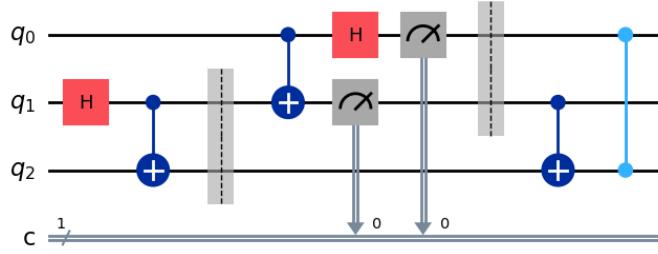


Figure 4.5: Teleportation Circuit

We can see from the figure that the measurement output from the Alice's qubits act as control bits for applying Z and X gate. If the first qubit is 1 then we apply the Z gate, if the second qubit is 1 then we apply the X gate. Based on the Alice's output we can decide whether to apply Thus the circuit shown in the figure 4.5, show that after Alice's measurement the bits act as control bits for the operations to be applied by Bob.

### 4.3.2 Superdense Coding

In Superdense coding we can send from one place to another two classical bits but using only one qubit. Alice and Bob create a bell state  $\beta_{00}$  and then distribute the qubits. Alice takes the first qubit and Bob takes the second qubit. Bob now moves light years away. Alice now wishes to send two classical bits to Bob. The protocol is as follows:

1. **Preparing Bell States:** Alice and Bob create a Bell state  $\beta_{00}$ . The state of the circuit after applying this state is the Bell state which is given as  $\beta_{00} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$ .

Now, say the first bit be  $|\beta_A\rangle$  which is the qubit that goes to Alice and the second bit be  $|\beta_B\rangle$  which is the qubit which goes to Bob.

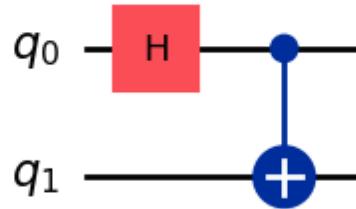


Figure 4.6: Creating Bell States

This is how the Bell state is created shown in the figure 4.6.

2. **Alice Encoding:** Alice applies the necessary operations to send the two classical bits to Bob. These operations are done on the first qubit which is with Alice as follows in the figure 4.7.

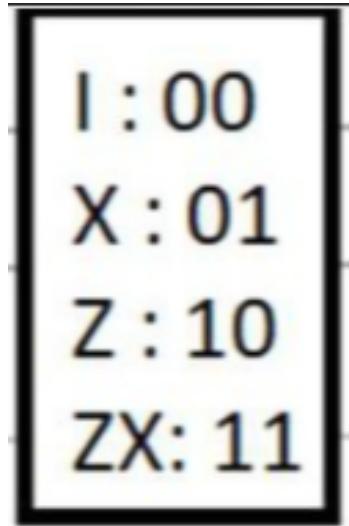


Figure 4.7: Alice's Encoding

- If Alice wishes to send 00, then she applies no operation.

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

- If Alice wishes to send 01, then she applies the Pauli - X gate on her portion of Bell state.

$$X \left( \frac{|00\rangle + |11\rangle}{\sqrt{2}} \right) = \frac{|10\rangle + |01\rangle}{\sqrt{2}}$$

- If Alice wishes to send 10, then she applies the Pauli-Z gate on the qubit.

$$Z \left( \frac{|00\rangle + |11\rangle}{\sqrt{2}} \right) = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$

- If Alice wishes to send 11, then she applies the Pauli-X and Pauli-Z gate on the qubit.

$$X \left( Z \left( \frac{|00\rangle + |11\rangle}{\sqrt{2}} \right) \right) = \frac{|10\rangle - |01\rangle}{\sqrt{2}}$$

3. **Bob Decoding:** Now Alice sends her portion of Bell state qubit to Bob. Bob then applies the necessary operations to get the two classical bits. Bob performs the following operations on the qubit he received from Alice: He applies CNOT gate with the first qubit (Qubit sent by Alice) as the control qubit and the second qubit as the target qubit. Then he applies Hadamard gate on the first qubit (Qubit sent by Alice). Then she performs the measurement. The measurement will give the two classical bits that Alice wished to send. This is as shown in the figure 4.8.

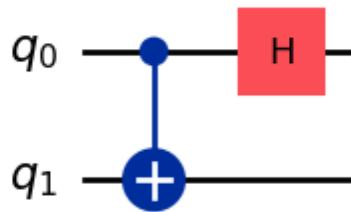


Figure 4.8: Bob's Decoding

- If Alice wished to send 00, then the state of the system is currently  $\frac{|00\rangle + |11\rangle}{\sqrt{2}}$ . Thus, after applying the CNOT gate,

$$CNOT \left( \frac{|00\rangle + |11\rangle}{\sqrt{2}} \right) = \frac{|00\rangle + |10\rangle}{\sqrt{2}}$$

Now, after applying the Hadamard Gate on the first qubit we get,

$$H \left( \frac{|00\rangle + |10\rangle}{\sqrt{2}} \right) = \frac{|00\rangle + |10\rangle + |00\rangle - |10\rangle}{2} = |00\rangle$$

Thus, on measurement, the two classical bits are 00.

- If Alice wished to send 01, then the state of the system is currently  $\frac{|10\rangle + |01\rangle}{\sqrt{2}}$ . Thus, after applying the CNOT gate,

$$CNOT \left( \frac{|10\rangle + |01\rangle}{\sqrt{2}} \right) = \frac{|11\rangle + |01\rangle}{\sqrt{2}}$$

Now, after applying the Hadamard Gate on the first qubit we get,

$$H \left( \frac{|11\rangle + |01\rangle}{\sqrt{2}} \right) = \frac{|01\rangle - |11\rangle + |01\rangle + |11\rangle}{2} = |01\rangle$$

- If Alice wished to send 10, then the state of the system is currently  $\frac{|00\rangle - |11\rangle}{\sqrt{2}}$ . Thus, after applying the CNOT gate,

$$CNOT \left( \frac{|00\rangle - |11\rangle}{\sqrt{2}} \right) = \frac{|00\rangle - |10\rangle}{\sqrt{2}}$$

Now, after applying the Hadamard Gate on the first qubit we get,

$$H \left( \frac{|00\rangle - |10\rangle}{\sqrt{2}} \right) = \frac{|00\rangle + |10\rangle - |00\rangle + |10\rangle}{2} = |10\rangle$$

- If Alice wished to send 11, then the state of the system is currently  $\frac{|10\rangle - |01\rangle}{\sqrt{2}}$ . Thus, after applying the CNOT gate,

$$CNOT \left( \frac{|10\rangle - |01\rangle}{\sqrt{2}} \right) = \frac{|11\rangle - |01\rangle}{\sqrt{2}}$$

Now, after applying the Hadamard Gate on the first qubit we get,

$$H \left( \frac{|11\rangle - |01\rangle}{\sqrt{2}} \right) = \frac{|01\rangle - |11\rangle - |01\rangle - |11\rangle}{2} = -|11\rangle$$

Thus, upon measurement the two classical bits are 11.

The final Superdense coding circuit is as shown in the figure 4.9.

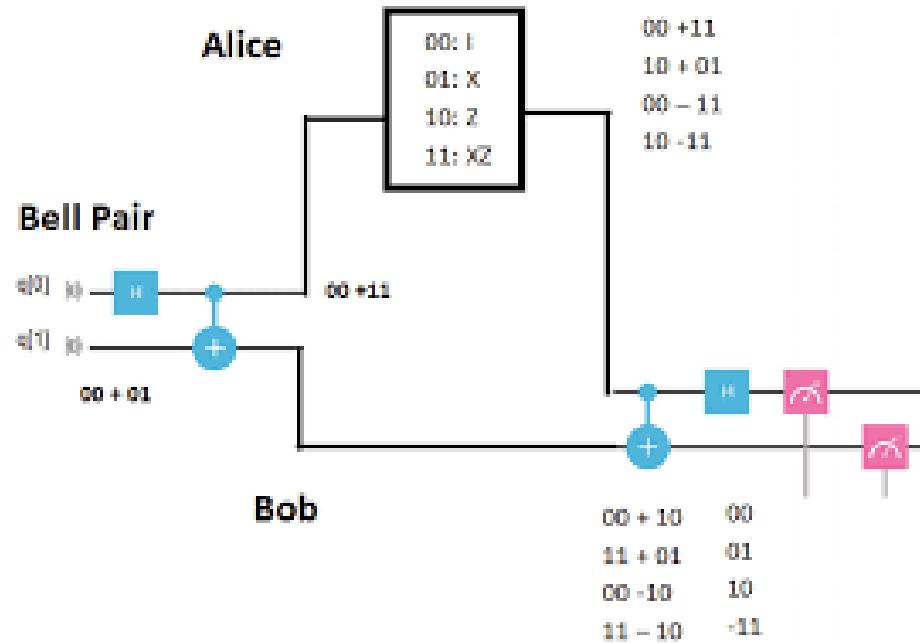


Figure 4.9: Superdense Coding Circuit

# Chapter 5

## Quantum Algorithms

Before reading this chapter it is advised to read the Appendix B. The Appendix B contains the theoretical foundation of classical circuit which would be later on explored in depth for understanding quantum complexity theory.

This chapter contains the prerequisites required before jumping into Quantum Algorithms in order to understand the various aspects of a Quantum Algorithm like their complexity, oracles and their working, depth and width and many more. The following are the Quantum Algorithms discussed in this book in the subsequent chapters:

1. Elitzur-Vaidman Bomb Detection Algorithm
2. Deutsch's Algorithm
3. Deutsch-Jozsa Algorithm
4. Bernstein-Vazirani Algorithm
5. Simon's Algorithm
6. Grover's Algorithm
7. Quantum Fourier Transform
8. Quantum Phase Estimation
9. Shor's Algorithm

## 5.1 Classical Circuits simulating Quantum Circuits

Classical circuits can simulate quantum circuits but the only known constructions have exponential overhead.

**Theorem 5.1.1.** *A quantum circuit of size  $s$  acting on  $n$  qubits can be simulated by a classical circuit of size  $\mathcal{O}(sn^22^n)$ .*

*Proof.* The idea of the simulation is quite simple. An  $n$ -qubit state is a  $2^n$  dimensional vector  $\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$ . Store the  $2^n$  in an array of size  $2^n$ , each with  $n$ -bits precision (which means accuracy within  $2^{-n}$ ) as shown in the figure 5.1.

$\alpha_{000}$
$\alpha_{001}$
$\alpha_{010}$
$\vdots$
$\alpha_{111}$

Figure 5.1:  $2^n$  dimensional array storing the amplitudes of the state  $\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$

The initial state is a computational basis state. Each gate corresponds to  $2^n \times 2^n$  matrix and the effect of the gate is to multiply the state vector by that matrix. In fact that matrix will be sparse for 1-qubit and 2-qubit gates, with only a constant number of nonzero entries for each row and each column. Therefore, there are a constant number of arithmetic operations per entry of the array. let's allow  $\mathcal{O}(n^2)$  elementary gates for each such arithmetic operation (on  $n$ -bit precision numbers). The simulation cost is  $\mathcal{O}(n^22^n)$  for each gate in the circuit. We multiply that by the number of gates in the quantum circuit  $s$  to get a cost of  $\mathcal{O}(sn^22^n)$ . That's the gate cost to get the final state of the computation, just before the measurement.

For measurements, we compute the absolute value squared of each entry of the array. Again, that's  $2^n$  arithmetic operations. At this point, the entries in the array are the probabilities after the measurement.

The output of the quantum circuit is not the probabilities; it's a sample according to the distribution. First we sample the first bit of the output, the probability that this bit is 0 is the sum of the first half of the entries of the array; the probability that this bit is 1 is the sum of the second half. Once, we have the first bit, we can have a similar approach for the second bit, using conditional probabilities, conditioned on the value of the first bit, and so on for the other bits. this also costs  $\mathcal{O}(n^22^n)$  elementary classical gates.

Note that if we take the quantum circuit that results from Shor's algorithm and then simulate it by a classical circuit then simulate that quantum circuit by a classical circuit, the result is not very efficient. It's exponential and in fact it's worse than the best currently-known classical algorithm for factoring.

If we view the aforementioned simulation in the usual high-level language of algorithms, it is exponential space in addition to exponential time (because it stores the huge array). In fact there's a way to reduce the store to polynomial while maintaining exponential time.  $\square$

## 5.2 Quantum Computation

Two models exist to explain how a quantum computer can apply computational steps to its register of qubits: the quantum Turing machine and the Quantum Circuit Model. These models are equivalent, in the sense that they can simulate each other in polynomial time, assuming the circuits are appropriately "uniform". We only explain the circuit model here, which is more popular.

## 5.3 Linear Growth and Duhamel's Principle

Consider a Quantum Algorithm denoted by a Unitary  $U$  (since every transformation in Quantum Computing is Unitary, thus a quantum algorithm can be thought of as a black box which performs a Unitary on some qubits and outputs those qubits (note that the number of qubits in input = number of qubits in the output)). Now, we know that the product of Unitary Matrices is a Unitary matrix. Thus we can decompose the Unitary  $U$  into a product of Unitary matrices  $U = U_1 U_2 \dots U_k$ . **Assumption:** Suppose we can implement each  $U_i$  to precision  $\epsilon$  then the following *Hybrid Argument*

**Proposition 5.3.1.** *Hybrid Argument:* Given Unitaries  $U_1, \tilde{U}_1, \dots, U_K, \tilde{U}_K \in \mathbb{C}^{N \times N}$  satisfying,

$$\|U_i - \tilde{U}_i\|_2 \leq \epsilon \quad \forall i = 1, \dots, K$$

we have,

$$\|U_K \dots U_2 U_1 - \tilde{U}_K \dots \tilde{U}_2 \tilde{U}_1\| \leq K\epsilon$$

*Proof.* Using a telescoping series, we can write:

$$\begin{aligned} U_K \dots U_2 U_1 - \tilde{U}_K \dots \tilde{U}_1 &= U_K \dots U_2 U_1 - U_K \dots U_2 \tilde{U}_1 + U_K \dots U_2 \tilde{U}_1 \\ &\quad - U_K \dots \tilde{U}_2 \tilde{U}_1 + \dots + U_K U_{K-1} \dots \tilde{U}_1 - \tilde{U}_K \dots \tilde{U}_1 \\ &= U_K \dots U_2 (U_1 - \tilde{U}_1) + U_K \dots U_3 (U_2 - \tilde{U}_2) \tilde{U}_1 + \dots \\ &\quad + U_K (U_{K-1} - \tilde{U}_{K-1}) \dots \tilde{U}_1 + (U_K - \tilde{U}_K) \dots \tilde{U}_1 \end{aligned}$$

Note that here we define the norm of a matrix as operator (spectral) norm  $\|A\| = \max_{\|x\|_2=1} \|Ax\|_2$ . Now, using the property that,  $\|A+B\| \leq \|A\| + \|B\|$  for matrices, we get:

$$\begin{aligned} \|U_K \dots U_2 U_1 - \tilde{U}_K \dots \tilde{U}_2 \tilde{U}_1\| &\leq \|U_K \dots U_2 (U_1 - \tilde{U}_1)\| \\ &\quad + \|U_K \dots U_3 (U_2 - \tilde{U}_2) \tilde{U}_1\| + \dots + \|U_K (U_{K-1} - \tilde{U}_{K-1}) \dots \tilde{U}_1\| \\ &\quad + \|(U_K - \tilde{U}_K) \dots \tilde{U}_1\| \end{aligned}$$

Now using the property that,  $\|AB\| \leq \|A\| \|B\|$  for matrices and using the property that the  $\|A\| = 1$  for a Unitary matrix since Unitary matrix preserves norm of the vector and only rotates it, Thus, we get:

$$\begin{aligned} \|U_K \dots U_2 U_1 - \tilde{U}_K \dots \tilde{U}_2 \tilde{U}_1\| &\leq \|U_K \dots U_2\| \|U_1 - \tilde{U}_1\| \\ &\quad + \|U_K \dots U_3\| \|U_2 - \tilde{U}_2\| \|\tilde{U}_1\| + \dots + \|U_K\| \|U_{K-1} - \tilde{U}_{K-1}\| \dots \|\tilde{U}_1\| \\ &\quad + \|U_K - \tilde{U}_K\| \dots \|\tilde{U}_1\| \\ &\leq \|U_1 - \tilde{U}_1\| + \|U_2 - \tilde{U}_2\| \dots \|U_K - \tilde{U}_K\| \\ &\leq \sum_{i=1}^K \|U_i - \tilde{U}_i\| \\ &\leq K\epsilon \end{aligned}$$

Thus, we say that if we can implement each local unitary to precision  $\epsilon$ , the global error grows at most linearly with respect to the number of gates and is bounded by  $K\epsilon$ . The above proof can be seen as a discrete analogue of the variation of constants method (also called Duhamel's principle).  $\square$

**Proposition 5.3.2.** (Duhamel's principle for Hamiltonian Simulation) Let  $U(t), \tilde{U}(t) \in \mathbb{C}^{N \times N}$  satisfy

$$\iota\partial_t U(t) = HU(t), \quad \iota\partial_t \tilde{U}(t) = H\tilde{U}(t) + B(t), \quad U(0) = \tilde{U}(0) = I$$

where  $H \in \mathbb{C}^{N \times N}$  is a Hermitian matrix and  $B(t) \in \mathbb{C}^{N \times N}$  is an arbitrary matrix. Then

$$\tilde{U}(t) = U(t) - \iota \int_0^t U(t-s)B(s)ds$$

and

$$\|\tilde{U}(t) - U(t)\| \leq \int_0^t \|B(s)\| ds$$

*Proof.* Substituting the equation,

$$\tilde{U}(t) = U(t) - \iota \int_0^t U(t-s)B(s)ds$$

into the equation,

$$\iota \partial_t \tilde{U}(t) = H\tilde{U}(t) + B(t)$$

we see that it satisfies. Now consider a special case where  $B(t) = E(t)\tilde{U}(t)$ , then

$$\tilde{U}(t) = U(t) - \iota \int_0^t U(t-s)E(s)\tilde{U}(s)ds$$

Now, taking the norm of the above equation we get:

$$\|\tilde{U}(t) - U(t)\| \leq \int_0^t \|E(s)\| ds$$

This is a direct analogue o the hybrid argument in the continuous setting.  $\square$

## 5.4 Universality of various sets of elementary gates

Which set of elementary gates should we allow? There are several reasonable choices.

1. The set of all 1-qubit operations together with the 2-qubit CNOT gate is universal, meaning that any other unitary transformation can be built from these gates. Allowing all 1-qubit gates is not very realisitc from an implementation point of view, as there are continuously many of them, and we cannot expect experimentalists to implement gates to infinite precision. However, the model is usually restricted, only allowing a small finite set of 1-qubit gates from which all other 1-qubit gates can be efficiently approximated.

2. The set consists of CNOT, Hadamard, and the phase-gate  $T = R_{\pi/4}$  is universal in the sense of approximation, meaning that any other unitary can be arbitrarily well approximated using circuits of only these gates. the *Solovay-Kitaev theorem* says that this approximation is quite efficient: we can approximate any gate on 1 or 2 qubits upto error  $\epsilon$  using a number of gates (from our small set) that is only  $\text{polylog}(1/\epsilon)$ , i.e., polynomial in the logarithm of  $1/\epsilon$ ; in particular, simulating arbitrary gates upto exponentially small error costs only a polynomial overhead. It is often convenient to restrict to real numbers and use an even smaller set of gates:
3. The set of hadamard and Toffoli (CCNOT) is universal for all unitaries with real entries in the sense of approximation, meaning that any unitary with only real entries can be arbitrarily well approximated using circuits of only these gates.

## 5.5 Universal gate sets and Reversible Computation

Universal gate sets are the set of gates which can be used to represent any gate as a combination of the gates from the set. In case of Classical Computation, the NAND gate is a universal gate set. The "NOR" gate is also a universal gate set.

**In case of Quantum Computations, any Unitary operator on n qubits can be implemented using 1- and 2- qubit gates.** A set of universal quantum gates  $S$  is universal if given any unitary operator  $U$  and desired precision  $\epsilon$  we can find  $U_1, \dots, U_m \in S$  such that

$$\|U - U_m \dots U_1\| \leq \epsilon$$

Some of the possible choices of the universal gate sets are  $\{\text{H}, \text{T}, \text{CNOT}\}$ ,  $\{\text{H}, \text{Toffoli}\}$

**Theorem 5.5.1.** (*Solovay-Kitaev Theorem*) Let  $\mathcal{S}, \mathcal{T}$  be two universal gate sets that are closed under inverses. Then any  $m$ -gate circuit using the gate set  $S$  can be implemented to precision  $\epsilon$  using a circuit of  $\mathcal{O}(m \cdot \text{polylog}(m/\epsilon))$  gates from the gate set  $T$ , and there is a classical algorithm for finding this circuit in time  $\mathcal{O}(m \cdot \text{polylog}(m\epsilon))$ . (Here *polylog* is the polynomial of the logarithm of the argument). **All choices of Universal gates are equivalent**

**Any classical circuit can also be asymptotically efficiently implemented using a quantum circuit.** (Classical gates can be efficiently simulated using quantum circuits.)

Step1: Note that NAND and other classical gates (such as AND, OR, etc..) are not reversible gates! Thus, we need to convert the classical gates to reversible gates. Any irreversible classical gate  $x \rightarrow f(x)$  can be converted to a reversible gate by adding an extra bit called the "garbage bit" and then applying the reversible gate. The reversible gate is then defined as  $x, y \rightarrow x, f(x) \oplus y$ . Thus, in particular we have  $(x, 0) \rightarrow (x, f(x))$ . the key idea here is to store all the intermediate steps of the computation. On the quantum computer, storing all the intermediate computational steps is problematic:

- Waste of Quantum Resources
- Interediate bits stored in some extra qubits are still entangled to the quantum state of interest. (If environment interferes with the intermeditate result, then the quantum state of interest also gets affected.)

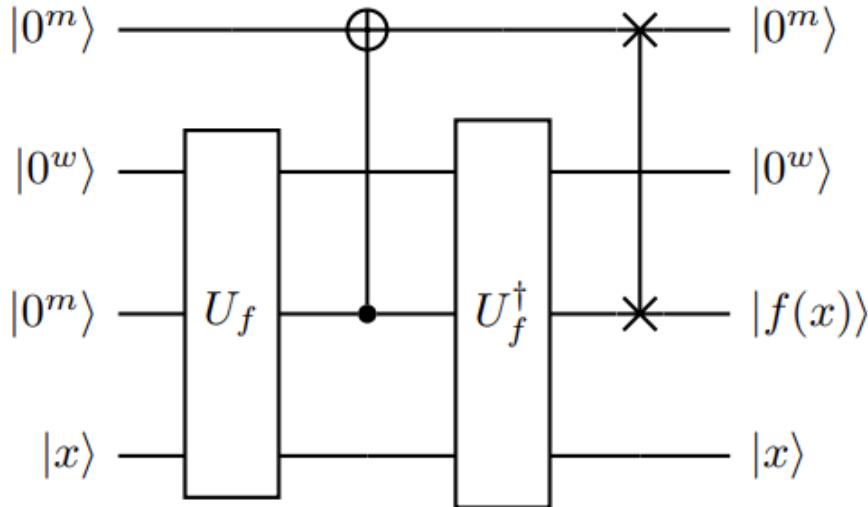


Figure 5.2: Uncomputation of a classical gate

Step 2: **Uncomputation:** To implement a boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  assume that there is an oracle.

$$|0^m\rangle |x\rangle \rightarrow |f(x)\rangle |x\rangle$$

where  $|0^m\rangle$  comes from a m-qubit output register. The oracle is often further implemented with the help of a working register (aka garbage register - they are ancilla registers which can be freed after uncomputation) such that

$$U_f : |0^w\rangle |0^m\rangle |x\rangle \rightarrow |g(x)\rangle |f(x)\rangle |x\rangle$$

From no deletion theorem, there is no generic unitary operator that can set a black box-state to  $|0^w\rangle$ . In order to set the working register back to  $|0^w\rangle$  while keeping the input and output state, we introduce another set of m-qubit ancillary registers initialised as  $|0^m\rangle$ . Then we use n-qubit CNOT controlled on the output register and obtain

$$|0^m\rangle |g(x)\rangle |f(x)\rangle |x\rangle \rightarrow |f(x)\rangle |g(s)\rangle |f(x)\rangle |x\rangle$$

Note that the multi-qubit CNOT gate only performs the classical copying operation in the computational basis and thus does not violate the *No-Cloning Theorem*. Now we know,  $U_f^\dagger = U_f^{-1}$ ,

$$(I_m \otimes U_f^\dagger) |f(x)\rangle |g(x)\rangle |f(x)\rangle |x\rangle = |f(x)\rangle |g(x)\rangle |f(x)\rangle |x\rangle = |f(x)\rangle |0^w\rangle |0^m\rangle |x\rangle$$

The, we apply a SWAP operation on the ancilla and output registers to obtain

$$|f(x)\rangle |0^w\rangle |0^m\rangle |x\rangle \rightarrow |0^m\rangle |0^w\rangle |f(x)\rangle |g(x)\rangle$$

Thus, not the ancilla and the working register are set to the initial state. They are no longer entangled to the input or output register and can be reused for other purposes. This is called **Uncomputation**.

(Discarding Working Registers). Thus after the uncomputation step the first two registers as shown are uncahnged before and after the application of the circuit, though they are changed during the intermediate steps. Therefore 5.2 effectively implements a unitary operator

$$(I_{m+w} \otimes V_f) |0^m\rangle |0^w\rangle |0^m\rangle |x\rangle = |0^m\rangle |0^w\rangle |f(x)\rangle |x\rangle$$

or

$$V_f |0^m\rangle |x\rangle = |f(x)\rangle |x\rangle$$

NOte hter the definition of  $V_f$  all the working registers have been discarded allowing us to simplify notation and focus on the essence of Quantum Algorithm.

Thus, using the technique of uncomputation, if the map  $x \rightarrow f(x)$  can be efficiently implemented on a classical computer, then we can implement the map efficiently on a quantum computer as well. All reversible single-bit and two-bit classical gates can be implemented using single-bit and

two-bit quantum gates. Thus, the reversible map can be made into a unitary operation as

$$U_f : |x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$$

This proves that quantum computers are at least as powerful as classical computers.  $U_f$  can be applied to any superposition of the states in the computational basis. Thus, the quantum computer can perform the computation on all possible inputs simultaneously.

$$U_f : \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x, 0\rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x, f(x)\rangle$$

**Note:** This does not mean that we can efficiently implement the map  $|x\rangle \rightarrow |f(x)\rangle$ . If  $f$  is a bijection, and we have access to the inverse of the reversible circuit for computing  $f^{-1}$  then we may use the technique of uncomputation to implement such a map.

## 5.6 Quantum Circuits simulating Classical Circuits

It is natural to ask how the classical model of computation compares with the quantum model of computation. Can quantum circuits simulate classical circuits? Can classical circuits simulate quantum circuits? How efficient are these simulations?

For quantum circuits to simulate classical circuits, the Pauli X gate (that maps  $|0\rangle$  to  $|1\rangle$  and  $|1\rangle$  to  $|0\rangle$ ) can be viewed as a NOT gate, but what quantum gate corresponds to the AND gate? None of the operations that we have considered so far has any resemblance to the AND gate. The Toffoli gate makes such a connection. the 3-qubit Toffoli gate will be useful for simulating AND gates. It's denoted this way, like a CNOT gate, but with an additional control qubit as shown in figure 5.3.

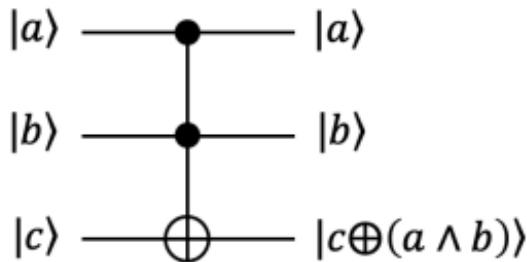


Figure 5.3: Toffoli gate acting on computational basis states ( $a, b, c \in \{0, 1\}$ )

On computational basis states  $|a\rangle$ ,  $|b\rangle$ , and  $|c\rangle$  it flips the third qubit if and only if both of the control qubits are in the state  $|1\rangle$ ; otherwise it does nothing. The  $8 \times 8$  unitary matrix for this gate is

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

For arbitrary 3-qubit states, that are not necessarily computational basis states, the action of the gate on the state is to multiply the state vector by this  $8 \times 8$  matrix. The Toffoli gate is sometimes called the controlled-controlled-NOT gate or controlled-CNOT gate.

**Theorem 5.6.1.** *Any classical circuit of size  $s$  can be simulated by a quantum circuit of size  $\mathcal{O}(s)$ , where the gates used are Pauli X, CNOT, and Toffoli.*

*Proof.* Recall that in classical computation NAND gate is a universal gate set (or  $\{\text{NOT}, \text{AND}\} = \{\text{NAND}\}$ ) constitute a universal gate set, NOT (AND)=NAND). Thus, if we are able to simulate NOT and AND gates on a quantum computer then we can convert any classical circuit into a circuit made up of only NOT and AND and then to a corresponding quantum circuit by replacing the classical NOT and AND gates with their corresponding quantum gates (Pauli X and Toffoli and CNOT gate for fanout operation respectively) counterparts. We use Toffoli gates to simulate AND gates as illustrated in figure 5.4.

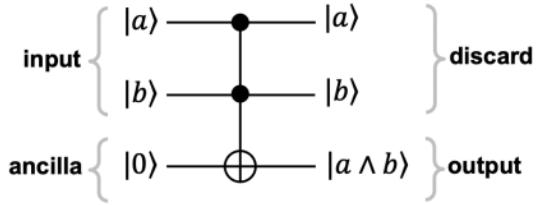


Figure 5.4: Using a Toffoli gate to simulate classical AND gate

Bits are represented as computational basis states in the natural way (bits  $a, b \in \{0, 1\}$  are represented by  $|a\rangle$  and  $|b\rangle$ ). To compute the AND of  $a$  and  $b$ , third ancilla

qubit in state  $|0\rangle$  is added and then a Toffoli gate is applied as shown in figure 5.4. This causes the state of the ancilla qubit to become  $keta \wedge b$ . The first two qubits can be discarded, or just ignored for the rest of the computation. That's how an AND gate can be simulated.

To simulate NOT gates, we can use the Pauli X gate. Finally, we can explicitly simulate a fan-out gate by adding an ancilla in state  $|0\rangle$  and apply a CNOT gate, as shown in figure 5.5.

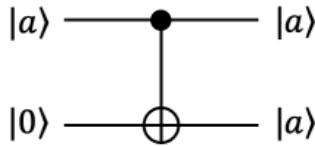


Figure 5.5: Using a CNOT gate to simulate a classical fan-out gate

Note that we defined quantum circuits to be in terms of 1-qubit and 2-qubit gates, but our simulation of classical circuits used 3-qubit gates: the Toffoli gate. We can remedy this by simulating each Toffoli gate by a series of 2-qubit gates. Let's show how to do this. To start, we will make use of a  $2 \times 2$  unitary matrix with the property that if you square it you get Pauli X. Since X is essentially the NOT gate, this matrix is sometimes referred to as a "square root of NOT". Note that, in classical information, there is no square root of NOT, so the quantum square root of NOT can be regarded as a curiosity. Let's refer to this matrix as V. Henceforth, assume that we have such a matrix V in hand. From this, we can define a controlled-V gate. Also, we can define a controlled- $V^*$  gate. Now, consider the following circuit, consisting of 2-qubit gates.

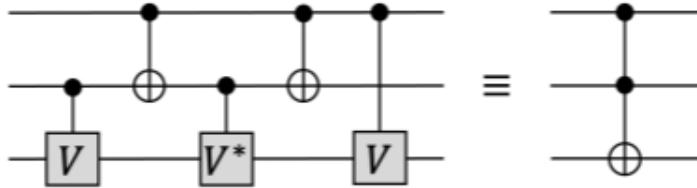


Figure 5.6: Simulating a Toffoli gate in terms of 2-qubit gates

This computes Toffoli gate. we can work out the  $8 \times 8$  matrix corresponding to each gate and then multiply the five matrices and see if the product is the matrix for the toffoli gate.

A second approach is to verify that the circuits are equivalent in the case where the first two qubits are in computational basis states  $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ .  $\square$

Thinking about this way, we can gain some intuition about why the circuit works. in the future if you need to design a quantum circuit to achieve something, such intuition can be helpful.

Some classical algorithms make use of random number generators and we have not captured this in our definition of classical circuits. Say we allow our classical algorithms to access random bits, that are zero with probability  $\frac{1}{2}$  and one with probability  $\frac{1}{2}$ . Can we simulate such random bits with quantum circuits? This is actually quite easy, since constructing a  $|+\rangle$  state and measuring it yields a random bit. But this entails an intermediate measurement. our quantum circuits will not look like ones we defined earlier, where all the measurements are at the end. Can we simulate random bits without the intermediate measurements? the answer is yes, by the following construction

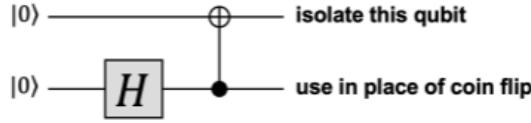


Figure 5.7: Simulating random bits without using measurements

Instead of measuring the qubit in the  $|+\rangle$  state, we apply a CNOT gate to a target qubit (an ancilla) in the state  $|0\rangle$ . Then we ignore that ancilla qubit for the rest of the computation. The final probabilities when the circuit is measured at the end will be exactly the same as if a random bit had been inserted at that place in the computation.

Using our decomposition of the Toffoli gate into 2-qubit gates and our results about simulating classical randomness, we can strengthen the above theorem to the following:

**Corollary 5.6.2.** *Any classical probabilistic circuit of size  $s$  can be simulated by a quantum circuit of size  $\mathcal{O}(s)$ , consisting of 1-qubit and 2-qubit gates.*

## 5.7 Computational Complexity Classes

On theoretical computer science, there are various taxonomies of computational problems according to their computational difficulty. The below shown are a few of these

and how the power of quantum computer fits in within this classification.

Consider all binary functions over the set of all binary strings (of the form  $f : \{0, 1\}^* \rightarrow \{0, 1\}$ ). The input is a binary string of some length  $n$ , where  $n$  can be anything. And the output is one bit. These are sometimes called decision problems since the answer is a binary decision, 0 or 1, yes or no, accept or reject. This is a common convention for reasons of simplicity. most problems that are not naturally decision problems can be reworked to be expressed as decision problems.

**Definition 5.7.1. P (polynomial time)** Solvable by  $\mathcal{O}(n^c)$ -size classical circuits (for some constant  $c$ ). technically, we require uniform circuit families, one for each input size.

**Definition 5.7.2. BPP (bounded-error probabilistic polynomial time)** Solvable by  $\mathcal{O}(n^c)$ -size probabilistic classical circuit whose worst-case error probability is  $\leq \frac{1}{4}$ .

Note that there is some arbitrariness in the error bound  $\leq \frac{1}{4}$ . Any polynomial-size circuit achieving this can be converted into another circuit whose error probability is  $\leq \epsilon$  by repeating the process  $\log(1/\epsilon)$  times and taking the majority value. using  $\frac{1}{4}$  is simple, though any constant below  $\frac{1}{2}$  would work.

In analogy too the classical class **BPP**, we will define **BQP** ("Bounded-error Quantum Polynomial time") as the class of languages that can efficiently be computed with success probability at least  $2/3$  by (a family of) quantum circuits whose size grows at most polynomially with the input length.

**Definition 5.7.3. BQP (bounded-error quantum polynomial time)** Solvable by  $\mathcal{O}(n^c)$ -size quantum circuits with worst-case error probability  $\leq \frac{1}{4}$ .

**Definition 5.7.4. EXP (exponential time)** Solvable by  $\mathcal{O}(2^n)$ -size classical circuits.

The following containments among these complexity classes are known:

$$\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{EXP}$$

## 5.8 Quantum Algorithms as Quantum Circuits

Now let's consider quantum circuits as shown in the figure 5.8.

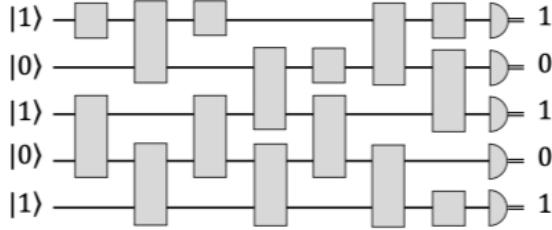


Figure 5.8: Generic form of a Quantum Circuit

The input gate is on the left, as computational basis states. Data flows from left to right as a series of unitary gates. Then measurement gates occur at the end, which yields the output of the computation.

Again, we can take various cost measures of quantum circuits, such as the number of gates, the depth, or the width. Let's focus on the number of gates. As with classical circuits, we need a notion of what an elementary gate is. For now let's consider the elementary gates to be the set of all 1-qubit and 2-qubit gates. We'll consider simpler gates later on.

Consider the factoring problem again. Peter Shor's famous factoring algorithm can be expressed as a quantum circuit for factoring that consists of  $O(n^2 \log n)$  gates - a polynomial number of gates! An efficient implementation of this algorithm would break several cryptosystems, whose security is based in the presumed computational hardness of factoring.

## 5.9 Fixed Point Number Representation and Classical Arithmetic operations

Any integer  $k \in [N] = \{0, 1, \dots, N-1\}$  where  $N = 2^n$  can be expressed as  $n$  n-bit string as  $k = k_{n-1} \dots, k_0$  with  $k_i \in \{0, 1\}$ . This is called binary representation of integer  $k$ .

$$k = \sum_{i \in [n]} k_i 2^i$$

The number  $k$  divided by  $2^m$  ( $0 \leq m \leq n$ ) (note that the decimal is shifted to be after  $k_m$ ).

$$a = \frac{k}{2^m} = \sum_{i \in [n]} k_i 2^{i-m} = (k_{n-1} \dots k_m.k_{m-1} \dots k_0)$$

when  $m=n$ ,

$$a = \frac{k}{2^n} = \sum_{i \in [n]} k_i 2^{i-n} = 0.k_{n-1} \dots k_0$$

$a = 0.k_{n-1} \dots k_0$  so that  $k_i$  is the  $i$ th decimal of  $a$  in the binary representation. Note that here  $0 \leq a \leq 1$ . The number  $(0.k_{n-1} \dots k_0)$  is called  $n$ -bit fixed point representation of  $a$ . Therefore to represent  $a$  the additive precision  $\epsilon$ ,  $n = \lceil \log_2(\epsilon) \rceil$  qubits are required. **We may add one extra bit to indicate its sign.**

Thus, we can perform classical arithmetic operations, such as  $(x, y) \rightarrow x + y$ ,  $(x, y) \rightarrow xy$ ,  $x \rightarrow x^\alpha$ ,  $x \rightarrow \cos \alpha$  etc. using reversible quantum circuits. **The number of ancilla qubits, and the number of elementary gates needed for implementing such circuits is  $O(\text{poly}(n))$ .** Thus, quantum computer is theoretically as powerful as classical computers, there is a very significant overhead in implementing reversible classical circuits on quantum devices, both in terms of the number of ancilla qubits and the circuit depth.

## 5.10 Fault tolerant Computation

We assumed that quantum operations can be perfectly performed and all errors come from either approximation errors at the mathematical level, or Monte Carlo errors in the readout process due to its probabilistic nature of the measurement process. Note that because of technical difficult quantum gates, and measurements involve significant errors on near-term quantum devices. **Threshold theorem:** If the noise in individual quantum gates is below a certain constant threshold (around  $10^{-4}$  or above) it is possible to perform an arbitrarily large computation with any desired precision. This requires quantum error correction protocols.

## 5.11 Complexity of Quantum Algorithms

Let  $n$  be the number of input bits. **Efficiency of Quantum Algorithm:** A Quantum Algorithm is efficient if the number of gates in the quantum circuit is  $O(\text{poly}(n))$ . Because of Probabilistic nature of measurements it is required for probability of correct answer to be higher. Generally,  $p > 2/3$  is considered to be a good probability of success or at least  $p > 1/2 + 1/\text{poly}(n)$ . Consider a case when Probability of getting a correct answer is  $p = 1/2 + \epsilon$  where  $\epsilon > 0$  is a constant. Then the best method of deciding the correct answer is based on the frequency of

the outcomes, the higher frequency outcome is the correct answer. But how many repetitions should be performed? Then the number of repetitions of the algorithm can be found using Chernoff bound as (say for binary outcome  $X_i = 0$  or  $X_i = 1$ ):

$$P\left(\sum X_i \leq n/2\right) \leq e^{-2n\epsilon^2}$$

where  $n$  is the number of times. Thus the error gets exponentially small. In Quantum Algorithms, Computational cost = Query Complexity. **Goal: Perform a given task using as few queries as possible to oracle ( $U_f$ )**. A Quantum Oracle (black box) is a Unitary operator that implements the function  $f$ .

**Example 5.11.1.** (Query access to boolean function)  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  be a boolean function and oracle be defined as:

$$U_f |x\rangle = (-1)^{f(x)} |x\rangle$$

where  $x \in \{0, 1\}^n$ . This is used in Grover's Algorithm. This is called phase kickback as  $f(x)$  is returned as a phase factor. Thus,  $U_f$  can be applied to superposition of the states in computational basis. Having query access to  $f(x)$  does not mean we know  $f(x)$ .

**Example 5.11.2.** (Partially specified oracles) Sometimes it is possible we might not be interested in entire  $U_f$  but only  $U_f$  applied to certain vectors say for example, as shown

$$U_f |0\rangle |x\rangle = |0\rangle A |x\rangle + |1\rangle B |x\rangle$$

$$U_f = \begin{bmatrix} A & * \\ B & * \end{bmatrix}$$

where  $A, B$  are arbitrary  $N \times N$  unitary matrices. This is called as partially specified oracle.

**Query complexity hides the implementation details of  $U_f$ .** For some cases, we can prove lower bounds on number of queries to solve a certain problems (example: Grover's Algorithm) but it's a difficult problem. Once we know number of elementary gates need to implement  $U_f$  we know gate complexity of the total algorithm. However, some queries can be (provably) difficult to implement, and there can be a large gap between the query complexity and gate complexity. Quantum Algorithm should not dominate the total gate complexity.

Circuit depth is the maximum number of gates on any path from input to output. It is equivalent of "Wall clock time" in classical computation. Quantum states can be

preserved only for a short amount of time = Coherence time. Thus circuit depth can be used to check whether the coherence time is exceeded by a Quantum Computer. **Quantum Algorithm should be depth efficient.** Thus, try to reduce the circuit time even if we need to run it multiple times.

A Quantum Algorithm consist of set of qubits (system registers - storing quantum states of interest and ancilla registers - auxiliary registers needed to implement the unitary operations acting on system registers) starting from an initial state , apply a series of one/two qubit states, and perform measurements, perform uncomputation whenever possible. Within ancilla register if a register can be freed by uncomputation then it is called a working register. Working register can be reused for other purposes. **We do not factor the cost of working registers into asymptotic cost analysis in the literature.**

## 5.12 Oracles

Before jumping into Quantum Algorithms we must understand the Oracles and their working. Consider a classical circuit representing a classical function  $f$ . Consider the classical function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  which takes  $n$  bits as input and gives  $m$  bits as output. **We can represent this classical function using a classical circuit.** (For the proof of this please refer to Appendix B.2) Thus, any classical boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  can be represented using a classical circuit as shown in the figure 5.9.

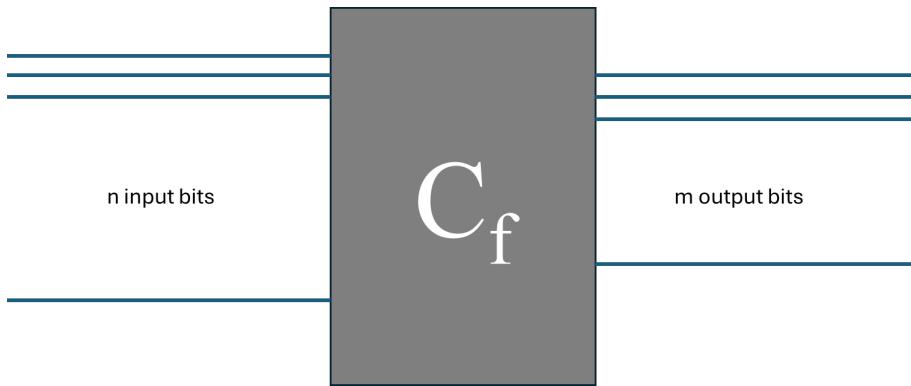


Figure 5.9: Classical Circuit

But we are interested in Quantum Circuits. Thus, we are required to convert this classical circuit into a Quantum Circuit. Recall, that a Quantum Circuit is a

sequence of Quantum Gates. Since all the operations using quantum gates done on qubits are unitary operations (thus, reversible), and thus the Quantum circuit can be written as a product of Unitary quantum gates which is a Unitary matrix (product of unitary matrices is a unitary matrix). Thus, the Quantum Circuit is itself a Unitary Gate and hence reversible. Thus, the Quantum circuits are reversible. Note that here the meaning of reversibility is that given the output of the circuit we can find the input of the circuit. This is because the Quantum Gates are Unitary and thus reversible. Thus, a Gate is said to be reversible if given the output of the gate we can find the input of the gate.

But, a classical circuit need not be necessarily reversible. Thus, we are required to convert the classical circuit into a reversible circuit in order to implement it on a Quantum Computer as a Quantum Circuit (Unitary Gates i.e. hence always reversible). Thus, we are required to transform the classical circuit in order to make it reversible in order to implement it as a Unitary matrix made up of Quantum Gates (reversible since  $U_f^\dagger = U_f^{-1}$ ).

This, can be done by transforming gates in a classical universal gate set to corresponding reversible quantum gates. Thus, then can we implement any classical circuit on a quantum computer. Note that a universal gate set is the set of gates which can be used to implement any classical circuit. There can be multiple gate sets which form the universal gate sets. For example, the  $\{NAND\}$  gate is a universal gate set,  $\{AND, NOT\}$  is another universal gate set. Thus, we can create any classical circuit using only the gates in the universal gate set. Now, if we are able to show that the gates in the universal gate set can be transformed into reversible quantum gates, then we can implement any classical circuit on a quantum computer using the corresponding reversible quantum gates. We can convert any classical circuit in terms of the universal gate sets and then replace the gates in the universal gate set with their corresponding reversible quantum gates to get the quantum circuit.

For the consideration let us take the Classical universal gate set  $\{AND, NOT\}$ . Here, NOT gate (refer Appendix B.1.1) is already reversible because if we know the output of the NOT gate we can find the input. For example, if the output of the NOT gate is 1 then we know that the input was 0 and if the output of the NOT gate is 0 then we know that the input was 1. This is because NOT is a single input single output gate (number of inputs=number of outputs) which negates the input. Thus, for the two inputs 0 and 1 we get the outputs 1 and 0 respectively. Thus, for finding the input from the given output we can simply negate that output to get the input (negation twice cancels out). Hence, the NOT gate is already a reversible gate. Its equivalent Quantum Gate is the Pauli-X gate which is also reversible. Thus, we can replace the NOT gate in the classical circuit with the Pauli-X gate in the Quantum

Circuit. Recall that the Pauli-X Gate is  $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$  and its action on  $|0\rangle$  and  $|1\rangle$  are  $X|0\rangle = |1\rangle$  and  $X|1\rangle = |0\rangle$  respectively. Thus, the Pauli-X gate is reversible and the quantum equivalent of the Classical NOT Gate.

Now, we are left with the AND gate. The AND gate is a two input single output gate i.e. It takes two inputs and gives one output. The output is 1 if and only if both the inputs are 1, else the output is 0 (refer Appendix B.1.2). Thus, for the case when the output is 0 the input could be either 00 or 01 or 10 but not 11. For the output of 1 we definitely know that the input was 11. But, for the output of 00 we cannot determine the input. Thus, the AND gate is not reversible. Thus, we are required to convert the AND gate into a reversible gate in order to implement it on a Quantum Computer. Thus, in order to create a reversible AND gate consider the control-Swap Gate (refer section 3.3.2). It takes three different inputs say  $|x\rangle$ ,  $|y\rangle$  and  $|z\rangle$ . Say  $x$  acts as the control bit and  $y$  and  $z$  are the target bits. This, is as shown in the figure 5.10

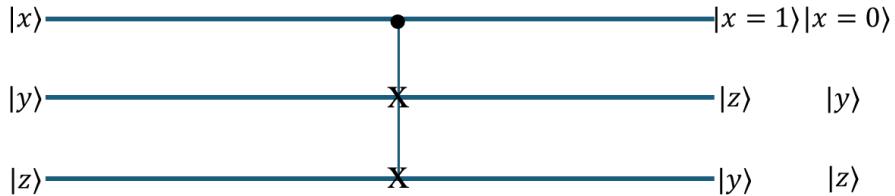


Figure 5.10: Control Swap Gate

Thus, if  $x = 1$  then the state of the system is swapped i.e.  $|y\rangle$  and  $|z\rangle$  are swapped, thus the output is  $|x\rangle, |z\rangle, |y\rangle$ . If  $x = 0$  then the state of the system remains the same  $|x\rangle, |y\rangle, |z\rangle$ .

Now in order to create reversible AND gate using the control-Swap gate we can do the following. Set the  $|z\rangle = |0\rangle$ . Then, the output of the control-Swap gate will be  $|x\rangle, |0\rangle, |y\rangle$  if  $x = 1$  and  $|x\rangle, |y\rangle, |0\rangle$  if  $x = 0$ . This, output can also be written in a more compact form as  $|x\rangle, (|0\rangle, |y\rangle), |x \wedge y\rangle$  where  $\wedge$  is the AND operation. The second qubit can be either  $|0\rangle$  or  $|y\rangle$  depending on the value of  $x$ . If  $x = 1$  then the second qubit will be  $|y\rangle$  and if  $x = 0$  then the second qubit will be  $|0\rangle$ . This, output of the second qubit is referred to as junk which is a function of  $x$  i.e.  $junk(x)$ . This is as shown in the figure 5.11.

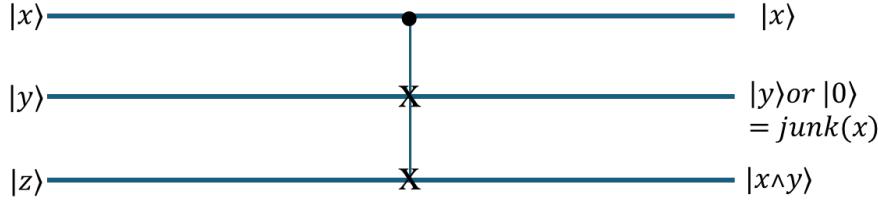


Figure 5.11: Reversible AND Gate

The truth table for the above circuit is as shown in the table 5.1.

$x$	$y$	0	$junk(x)$	$x \wedge y$
0	0	0	0	0
0	1	0	1	0
1	0	0	0	0
1	1	0	1	1

Table 5.1: Reversible AND Gate

Hence, we can clearly see that the output of the circuit is the same as the output of the AND gate. Thus, the above circuit is the reversible AND gate. Thus, using CSWAP gate as shown above we can convert the AND gate into a reversible gate.

Thus, we can convert any classical circuit into a reversible circuit by replacing the gates in the universal gate set with their corresponding reversible quantum gates. We replace NOT gate with the Pauli-X gate and the AND gate with the modified CSWAP Gate. Hence, any classical function can be implemented on a classical circuit (in terms of universal gate set  $\{AND, NOT\}$ ) which then can be converted to a reversible circuit by replacing the gates in the universal gate set with their corresponding reversible quantum gates (AND gate with Modified CSWAP and NOT gate with Pauli-X gate). Thus, for a general classical function with corresponding implementation in classical circuit as  $C_f$  we would have a corresponding quantum circuit  $U_f$  which is the reversible version of the classical circuit  $C_f$  thus, implementing the classical function  $f$  on a quantum circuit by making it reversible. Here, we have converted the classical function  $f$  to a reversible function  $\tilde{f}$  which has number of inputs equal to the number of outputs, produces the same output as the classical

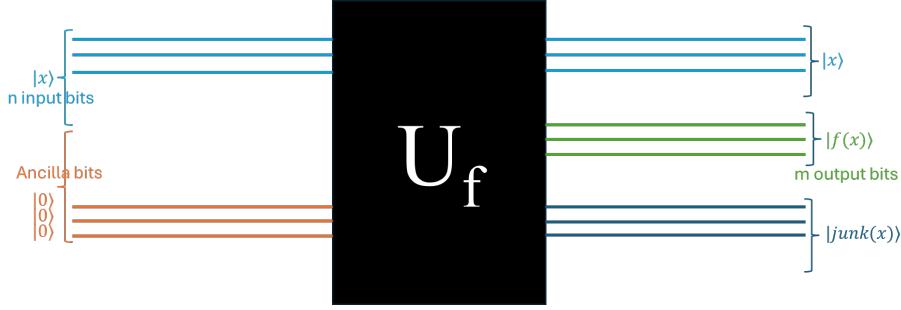


Figure 5.12: Quantum Circuit

function  $f$  along with the input and some junk values as shown in the figure 5.12. **We can implement any classical function on a quantum circuit.** The above figure shows the conversion of a classical circuit to a quantum circuit.

Thus, a classical circuit when converted to a quantum circuit is as shown in the figure 5.12, it takes input  $|x\rangle$  and along with it some ancilla bits (aka junk bits) and produces the output  $|x\rangle$ ,  $|y \oplus f(x)\rangle$  along with some junk bits  $junk(x)$ . We need to remove the junk since it consumes extra amount of qubits which is a waste of quantum resources and also the junk bits might get entangled with the output and any interference with environment may lead to decoherence or incorrect output and errors. Thus, we need to remove the junk bits. This is done using the following circuit 5.13

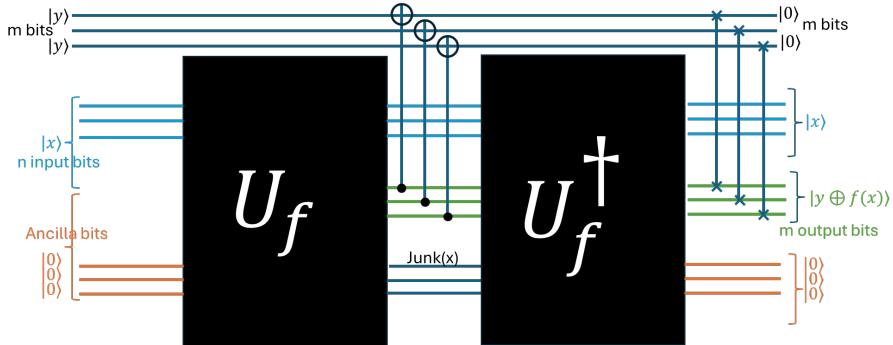


Figure 5.13: Quantum Oracle

Using this circuit it can be seen that the output of the entire circuit is  $|x\rangle$ ,  $|y \oplus f(x)\rangle$  and the junk bits are now  $|0\rangle$  so that they can be used for other purposes in the circuit ahead. The output from the first Quantum Circuit  $U_f$  is  $|x\rangle$ ,  $|y \oplus f(x)\rangle$ ,  $junk(x)$ . To remove the junk bits, it was done by first copying the output  $|f(x)\rangle$  to the other

(working bits) bits and then the output from the  $U_f$  circuit was passed through its inverse  $U_f^{-1} = U_f^\dagger$ . Thus, upon passing through its inverse the output will be  $|x\rangle, |0\rangle, |0\rangle$  since  $U_f^\dagger$  is a reverse circuit and hence will make the outputs back to the inputs. Then the bits are swapped between as shown in the figure to get the desired output of  $|x\rangle, |y \oplus f(x)\rangle, |0\rangle, |0\rangle$  with no junk. Thus, the output of the circuit is the input  $|x\rangle$  and the output  $|y \oplus f(x)\rangle$ . This entire thing is called an Oracle. Thus, we can imagine a Quantum Circuit which is an implementation of a classical function  $f$  as a black box which takes input  $|x\rangle, |y\rangle, |0\rangle$  and produces the output  $|x\rangle, |y \oplus f(x)\rangle, |0\rangle$ . For the sake of simplicity we generally ignore the  $|0\rangle$  in inputs and outputs.

Thus, whenever given a classical function  $f$ , we can make it as a reversible quantum function and thus, implement it as a quantum circuit using reversible quantum gates. This can be visualized as shown in the figure 5.14.

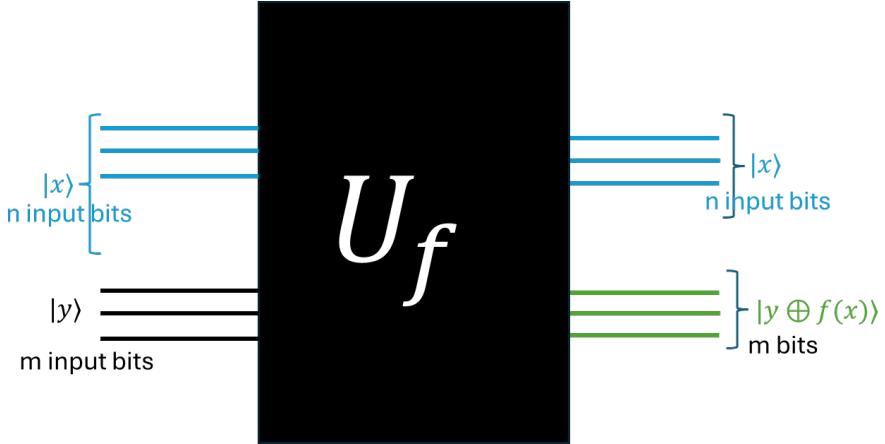


Figure 5.14: Oracle

## 5.13 Quantum Parallelism

One uniquely quantum-mechanical effect that we can use for building quantum algorithms is quantum parallelism. Suppose we have a classical algorithm that computes some function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ . Then we can build a quantum circuit  $U$  (consisting only of Toffoli gates) that maps  $|z\rangle|0\rangle \rightarrow |z\rangle|f(z)\rangle$  for every  $z \in \{0, 1\}^n$ . Now suppose we apply  $U$  to a superposition of all inputs  $z$  (which is easy to build

using  $n$  Hadamard transformations):

$$U \left( \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} |z\rangle |0\rangle \right) = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} |z\rangle |f(z)\rangle$$

We applied  $U$  just once, but the final superposition contains  $f(z)$  for all  $2^n b$  input values  $z$ ! however, by itself this is not very useful and does not give more than classical randomization, since observing the final superposition will give just one uniformly random  $|z\rangle |f(z)\rangle$  and all other information will be lost. as we will see below, quantum parallelism needs to be combined with the effects of interference and entanglement in order to get something that is better than classical.

# Chapter 6

## Elitzur-Vaidman Bomb Detection Algorithm

### 6.1 The Problem

We are given a device which can be a working Bomb or dud. The device takes a Quantum bit as input and produces an output. If the device is a bomb, then the device performs a measurement on that qubit. If the outcome of the measurement is 1, then the bomb explodes, else the bomb does not explode. If the device is dud, then the device does nothing i.e output is same as input. We wish to determine whether the device is a bomb or dud without exploding the bomb. The Elitzur-Vaidman Bomb Detection Algorithm is as follows:

#### 1. Approach 1: Naive Approach

We give the device naively the input of  $|0\rangle$  and  $|1\rangle$ .

- **Case 1: Device is a Bomb**

If we give device input of  $|0\rangle$  then the output will also be  $|0\rangle$  always, because the outcome of the measurement will be  $|0\rangle$  so the bomb doesn't explode and returns  $|0\rangle$ . If we give device input of  $|1\rangle$  then upon measurement the outcome will be  $|1\rangle$  and the bomb explodes always.

- **Case 2: Device is a Dud**

For the input of  $|0\rangle$  if the device is a dud then the output is  $|0\rangle$  since it does no operation on the input. For the input of  $|1\rangle$  if the device is a dud then the output is  $|1\rangle$  since it does no operation on the input.

To summarise the above approach, giving input  $|0\rangle$  is of no help as it provides

no information about the device since in both the cases if its a bomb or a dud the output will be  $|0\rangle$ . Upon giving input  $|1\rangle$  if the device is a dud then the output will be  $|1\rangle$  and if the device is a bomb then the bomb will explode. Thus, it tells us whether the device is a bomb or a dud but if its a bomb it will explode. Thus, this approach is not useful since we can never detect a working bomb.

## 2. Approach 2: Quantum Superposition Approach

Here we use a Hadamard Gate before and after the device. The circuit is as shown in the figure 6.1.

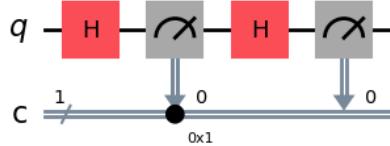


Figure 6.1: Elitzur-Vaidman Bomb Detection Algorithm

Here the first measurement is present if device is a bomb.

- **Case 1: Device is a Bomb**

If we give input of  $|0\rangle$  then upon passing through the hadamard gate it will be  $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ . As it passes through the device, it performs a measurement since its a bomb. Thus, the outcome of the measurement in computational basis will be  $|0\rangle$  with probability  $\frac{1}{2}$  and  $|1\rangle$  with probability  $\frac{1}{2}$ . Thus, in the cases where the outcome is  $|0\rangle$  the bomb will not explode and the output will then pass through another Hadamard gate which will give output as  $|+\rangle$ . Thus, upon final measurement we will get  $|0\rangle$  with probability  $\frac{1}{2}$  and  $|1\rangle$  with probability  $\frac{1}{2}$ . In the case where the measurement outcome is  $|1\rangle$  the bomb will explode.

- **Case 2: Device is a Dud**

If we give input of  $|0\rangle$  then the upon passing through the hadamard gate it will be  $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ . As it passes through the device, the device does nothing and the output will be  $|+\rangle$ . Then, we pass through the Hadamard gate which will output  $|0\rangle$  and upon final measurement we will get the output as  $|0\rangle$ .

To summarize, the above approach, suppose there are 1000 devices out of which probability of being a bomb is 50%, thus, 500 are bombs and 500 are duds. If we given input  $|0\rangle$  to all the devices. Then the output will be  $|0\rangle$  for all the duds. For the bombs, because of the measurement 50% will detect  $|1\rangle$  and will explode while the other 50% will not explode and output  $|0\rangle$ . Thus, out of 500 bombs, 250 will explode and the rest 250 won't explode. Then, the out of the bombs which didn't explode and which give outcomes as  $|0\rangle$ , it will pass through a hadamard gate whose output will be  $|+\rangle$  and thus upon final measurement we will get  $|0\rangle$  with probability  $\frac{1}{2}$  and  $|1\rangle$  with probability  $\frac{1}{2}$ . Thus, out of the 250 bombs which didn't explode 50% i.e. 125 of them will output  $|1\rangle$  and the other 50% i.e. 125 will output  $|0\rangle$ .

Thus, out of the 1000 devices we will get 250 bombs which will explode and out of the rest 750 devices we will get output of  $|0\rangle$  for 625 (500 duds + 125 bombs which gave output  $|0\rangle$ ) devices and output of  $|1\rangle$  for 125 devices. Now, we are specifically interested in the cases where the output was  $|1\rangle$  upon input  $|0\rangle$ . This is possible only in the case where the devices were bomb which didn't explode, since had it been a dud the output would have been  $|0\rangle$ . Thus the 125 cases where the output was  $|1\rangle$  upon input of  $|0\rangle$  are the bombs which didn't explode. Thus, out of the 500 bombs 250 of them exploded and we found 125 working bombs, and the other 125 bombs didn't explode but couldn't be identified among the duds. Thus, we found 25% of the working bombs without exploding them. Or, we can say that the we can find a working bomb with a probability of  $\frac{1}{8}$  out of all the given devices (or we can find a working bomb without exploding given that the device is a bomb with probability  $\frac{1}{4}$ ).

**What if the input had been  $|1\rangle$ ?:** The result will be the same as above. The only difference is that the output will be  $|1\rangle$  for the duds and the bombs which didn't explode. Some of the bombs which didn't explode will output  $|0\rangle$  and thus, can be detected as working bombs.

Thus, we can find a working bomb with a probability of  $\frac{1}{8}$  (i.e. without exploding) out of all the given devices (or we can find a working bomb without exploding given that the device is a bomb with probability  $\frac{1}{4}$ ). But, recall that in this case  $\frac{1}{2}$  (i.e. 250 bombs) of the bombs exploded and out of the other  $\frac{1}{2}$  (i.e. 250 bombs) which didn't explode we only found  $\frac{1}{2}$  (i.e. 125 bombs) of the working bombs, the rest  $\frac{1}{2}$  (i.e. 125 bombs) didn't explode but couldn't be identified either.

### 3. Elitzur-Vaidman Bomb Detection Algorithm:

Recall that we are measuring in orthonormal computational basis i.e.  $|0\rangle$  and  $|1\rangle$ . Consider the Single-Qubit gate  $R_\theta^Y$  which is defined as:

$$R_\theta^Y = \begin{bmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$$

where  $\theta$  is the angle of rotation. The  $R_\theta^Y$  gate is a rotation gate about the Y-axis. The  $R_\theta^Y$  gate is a unitary gate and thus preserves the norm of the vector as well as inner product. It is also a Hermitian gate. Thus,  $R_\theta^Y$  is a self-inverse gate.  $R_\theta^Y = (R_\theta^Y)^\dagger = (R_\theta^Y)^{-1}$ . Thus, the action of  $R_\theta^Y$  gate on the state  $|0\rangle$  is as follows:

$$R_\theta^Y |0\rangle = \begin{bmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos \frac{\theta}{2} \\ \sin \frac{\theta}{2} \end{bmatrix} = \cos \frac{\theta}{2} |0\rangle + \sin \frac{\theta}{2} |1\rangle$$

Thus, upon measurement the probability of getting  $|0\rangle$  is  $|\cos \frac{\theta}{2}|^2$  and the probability of getting  $|1\rangle$  is  $|\sin \frac{\theta}{2}|^2$ . Thus, the  $R_\theta^Y$  gate is a rotation gate about the Y-axis by an angle  $\theta$ . Now clearly, for very small  $\theta$ , the probability of getting  $|0\rangle$  is approximately  $\cos \frac{\theta}{2}$ . The circuit for Elitzur-Vaidman Bomb Detection Algorithm is as shown in the figure 6.2.

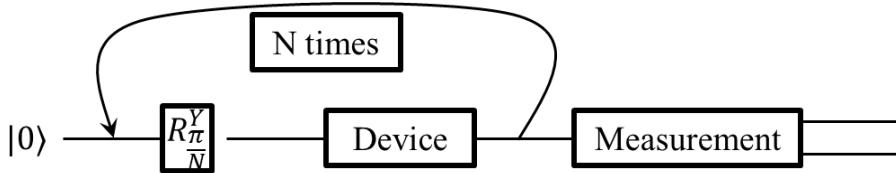


Figure 6.2: Elitzur-Vaidman Bomb Detection Algorithm

Thus, we pass  $|0\rangle$  through the  $R_\theta^Y$  gate, the output will be  $\cos \frac{\theta}{2} |0\rangle + \sin \frac{\theta}{2} |1\rangle$ .

- **Case 1:Device is a Dud**

Now since the device is Dud, it will do nothing and after the first iteration, the output will be  $\cos \frac{\pi}{2N} |0\rangle + \sin \frac{\pi}{2N} |1\rangle$ . Now, for the second iteration it will be passed through the  $R_\theta^Y$  gate again and the output will be:

$$\begin{bmatrix} \cos \frac{\pi}{2N} & -\sin \frac{\pi}{2N} \\ \sin \frac{\pi}{2N} & \cos \frac{\pi}{2N} \end{bmatrix} \begin{bmatrix} \cos \frac{\pi}{2N} \\ \sin \frac{\pi}{2N} \end{bmatrix} = \begin{bmatrix} \cos^2 \frac{\pi}{2N} - \sin^2 \frac{\pi}{2N} \\ 2 \cos \frac{\pi}{2N} \sin \frac{\pi}{2N} \end{bmatrix} = \cos \frac{\pi}{N} |0\rangle + \sin \frac{\pi}{N} |1\rangle$$

Thus, after the second iteration the output will be  $\cos \frac{\pi}{N} |0\rangle + \sin \frac{\pi}{N} |1\rangle$ . Thus, after the kth iteration the output will be  $\cos \frac{k\pi}{2N} |0\rangle + \sin \frac{k\pi}{2N} |1\rangle$ .

Thus, at the last Nth iteration it will be  $\cos \frac{\pi}{2} |0\rangle + \sin \frac{\pi}{2} |1\rangle = |1\rangle$ . Thus, upon the final measurement the output will be  $|1\rangle$ . Thus, for the case when the device is a dud we will get  $|1\rangle$  as the output.

- **Case 2: Device is a Bomb**

Now since the device is a bomb, it will perform a measurement. Thus, after passing the input  $|0\rangle$  through the rotation gate, the output will be  $\cos \frac{\pi}{2N} |0\rangle + \sin \frac{\pi}{2N} |1\rangle$ . Now since the device is a bomb, it performs a measurement and the measurement will be  $|0\rangle$  with probability  $\cos^2 \frac{\pi}{2N}$  and  $|1\rangle$  with probability  $\sin^2 \frac{\pi}{2N}$ . Now consider for very large N, the probability of getting  $|0\rangle$  is  $\lim_{N \rightarrow \infty} \cos^2 \frac{\pi}{2N} \approx \cos 0 \approx 1$  and the probability of getting  $|1\rangle$  is  $\lim_{N \rightarrow \infty} \sin^2 \frac{\pi}{2N} \approx \sin 0 \approx 0$ . Thus, the output will be  $|0\rangle$  with a very high probability and  $|1\rangle$  has almost zero probability for very large N. Thus, after the first iteration we will get  $|0\rangle$  as the output with very high probability and this is then again passed through the rotation gate for the next iteration. Now, the second iteration will be the same as the first iteration since the input here is again  $|0\rangle$  i.e.e same as in the first iteration and this will repeat for N iterations. After the N iterations the circuit will output  $|0\rangle$  provided the bomb doesn't explode even though the probability of the bomb exploding is very low. Thus, the output of the circuit will be  $|0\rangle$  for the case when the device is a bomb or it will explode with a very low probability. Note that the chances of the bomb exploding is inversely proportional to the value of N i.e. the number of times we run the circuit. Thus, the chances of the bomb exploding is very low for very large N. Hence, we get the output as  $|0\rangle$  for the case when the device is a bomb.

The probability of that the bomb explodes after first iteration is  $\sin^2 \frac{\pi}{2N}$  and the probability that the bomb explodes after N iterations is  $\sin^{2N} \frac{\pi}{2N}$  (this is because for the explosion of bomb is an independent event in each iteration and for independent events  $P(A \cap B \cap \dots \cap Z) = P(A) \cdot P(B|A) \dots P(Z|A, B, \dots, Z) = P(A) \cdot P(B) \dots P(Z)$ , thus the probability P(bomb exploding in 1st iteration and bomb exploding in second iteration and ... and bomb exploding in Nth iteration))=P(bomb exploding in first iteration)P(bomb exploding in second iteration)...P(bomb exploding in N iterations). For very large N  $\sin \frac{\pi}{2N} \approx \frac{\pi}{2N}$ . Hence, the probability of the bomb exploding after N iterations will be  $\left(\frac{\pi}{2N}\right)^{2N}$ .

N	Probability of Bomb Exploding
2	0.25
10	$7.702 \times 10^{-17}$
20	$6.105 \times 10^{-45}$
:	:
500	$1.409 \times 2^{-2503}$

Table 6.1: Probability of Bomb Exploding after N iterations

As can be seen in the table 6.1, the probability of the bomb exploding after N iterations drops rapidly for very large N. Thus, the Elitzur-Vaidman Bomb Detection Algorithm is a very efficient algorithm for detecting a working bomb without exploding it.

# Chapter 7

## Deutsch's Algorithm

This is one of the simple quantum algorithms in this section which is admittedly curiosities, rather than practical algorithms. It's hard to think of real-world applications that fit their frame-work, and where it worth the trouble to build a quantum device to solve thee problems. It is important to pay attention to the ideas and maneuvers that these quantum algorithms make. After these algorithms, we will be seeing increasingly sophisticated extensions of these maneuvers, that accomplish more dramatic algorithmic feats. This is a very different setting in which some advantage is gained by using quantum information over classical information.

### 7.1 The Problem Definition

**Definition 7.1.1.** Given a function  $f : \{0, 1\} \rightarrow \{0, 1\}$ , which takes a single bit as input and produces a single bit as output. The function  $f$  is said to be constant if  $f(0) = f(1)$  and  $f$  is said to be balanced if  $f(0) \neq f(1)$ . The goal is to determine whether or not  $f(0) = f(1)$  by making queries to  $f$ .

There are four possible functions as shown in the table 7.1. Out of these four possible cases we can see that two are constant and the other two are balanced functions.

Function	Input	Output
$f_1$	0	0
	1	0
$f_2$	0	1
	1	1

(a) Constant Function

Function	Input	Output
$f_3$	0	0
	1	1
$f_4$	0	1
	1	0

(b) Balanced Function

Table 7.1: Possible Functions

Given a function  $f$ , our aim is to determine whether the given function is constant or balanced with as minimum number of queries as possible to the function.

## 7.2 The Classical Solution

**On a Classical Computer:** Let's first consider classical queries necessary to solve Deutsch's problem. One query is not sufficient. To see why this is so, suppose that you make one query at some  $a \in \{0, 1\}$  to acquire  $f(a)$ . This gives absolutely no information about the other value,  $f(\neg a)$ . It is possible that  $f(\neg a) = f(a)$  and it is possible that  $f(\neg a) \neq f(a)$ . Therefore, two queries and clearly two queries are sufficient. In conclusion, to solve this problem on a classical computer we are required to make two function calls. If both the outputs are the same then the function is constant and if the outputs are different then the function is balanced. Thus, we require **two classical queries/function calls** to determine whether the function is constant or balanced. Thus, two evaluations of the functions are necessary and sufficient to answer the question. If only one evaluation is permitted, the function could still be either constant or balanced regardless of the input and output obtained.

One may wonder whether the number of reversible classical queries is different. In fact, reversible classical query at  $(a, b)$  provide exactly the same amount of information as a simple classical queries of at  $a$ . The output of the reversible query at  $(a, b)$  is  $(a, b \oplus f(a))$ , and note that  $(a, b)$  are already known. Therefore, there are only two possibilities of interest for the output:

$$b \oplus f(a) = b, \text{ which occurs if and only if } f(a) = 0$$

$$b \oplus f(a) \neq b, \text{ which occurs if and only if } f(a) = 1$$

Therefore, even with reversible classical queries, one query is not sufficient. An

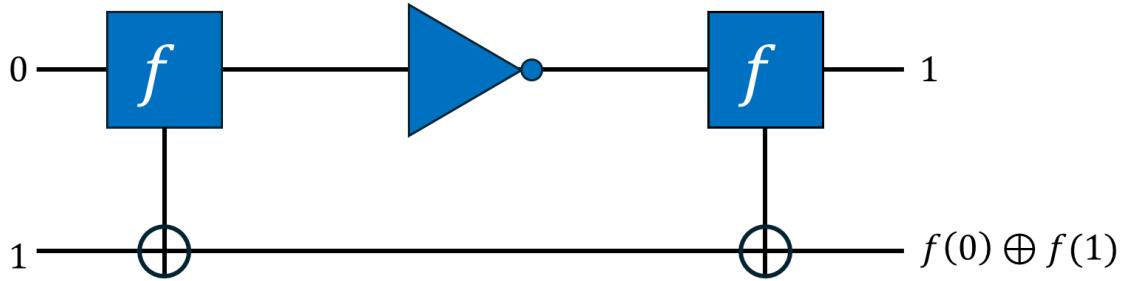


Figure 7.1: Classical Deutsch Circuit

algorithm solving Deutsch's problem with two reversible classical queries looks like as shown in the following classical circuit 7.1.

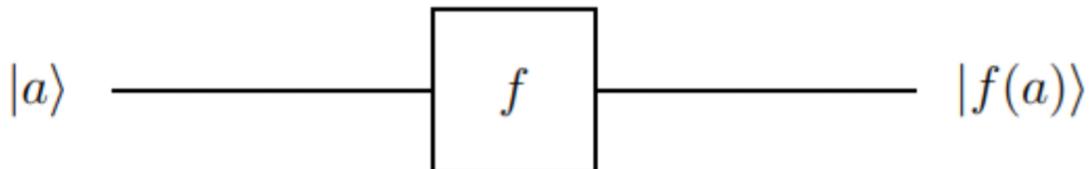
The first query XORs the value of  $f(0)$  to the second bit. Then the first bit is flipped to 1, so the second query XORs the value of  $f(1)$  to the second bit. At the end, the second bit contains the value of  $f(0) \oplus f(1)$ , which is the solution to Deutsch's problem.

There is an obvious 2-query quantum algorithm that solves the Deutsch's problem just like the circuit shown in the figure 7.1.

### 7.3 The Quantum Solution: Deutsch's Algorithm

**On a Quantum Computer:** To solve this problem on a Quantum Computer using Deutsch's Algorithm we are required to make only one function call. This speedup is because of the superposition.

Now when we consider the same question in the context of quantum information. We need to change the question slightly, however, in order for it to fit into the model of quantum information that we are considering. More specifically, we must insist that the action of the device corresponds to a unitary transformation. It is therefore not sufficient to consider the black box to be a one qubit action as follows:



For example, if  $f = f_3$  then this gate would correspond to the matrix

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

which is not unitary.

Instead, for any function  $f : \{0, 1\} \rightarrow \{0, 1\}$  we define a 2-qubit quantum gate  $B_f$  as follows:

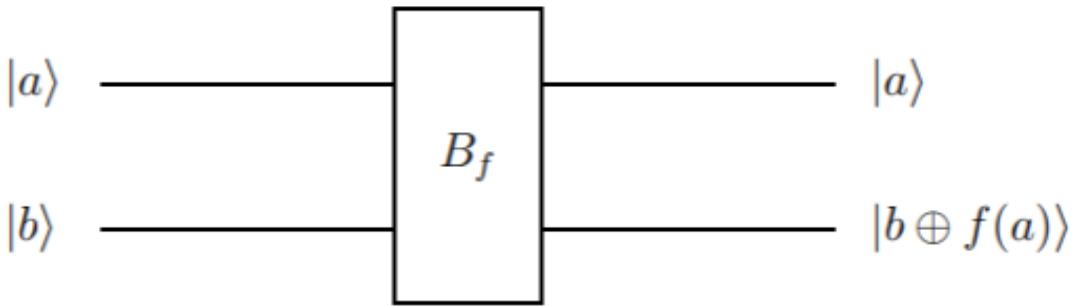


Figure 7.2: Oracle

It can be verified that the corresponding matrix is unitary for any function  $f$ . For example, if  $f = f_4$  from the table 7.1, then  $f(0_1)$  and  $f_1) = 0$ , so the corresponding matrix is

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

For any function  $f$ , the matrix corresponding to  $B_f$  will always be a permutation matrix, meaning that all of the entries are 0 or 1 and every row and every column has exactly one 1 in it. Permutation matrices are always unitary.

In general, if

$$f : \{0, 1\}^n \rightarrow \{0, 1\}^m$$

is any function (for any positive integers  $n$  and  $m$ ). The associated quantum transformation  $B_f$  will be defined by

$$B_f |x> |y> = |x> |y ⊕ f(x)>$$

(where  $\oplus$  denotes the bitwise exclusive OR). The associated matrix will always be a permutation matrix, and is therefore unitary. Note that we generally assume the

second input to the oracle to be  $|0\rangle$  so this simplifies the oracle to as follows:

$$B_f |x\rangle |0\rangle = |x\rangle |0 \oplus f(x)\rangle = |x\rangle |f(x)\rangle$$

The circuit for Deutsch's Algorithm is as shown in the figure 7.3.

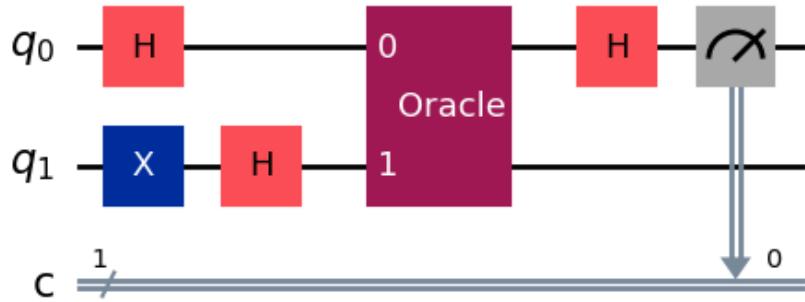


Figure 7.3: Deutsch's Algorithm

Consider the oracle as shown in the figure 7.3. The oracle is a black box which takes inputs  $|x\rangle$  and  $|y\rangle$  as inputs and outputs  $|x\rangle$  and  $|y \oplus f(x)\rangle$  as outputs. It is a unitary operator (denoted by  $U_f$ ) and implements the functions  $f$ .

### 7.3.1 Idea of the Algorithm

The primary limitation of the classical algorithm is that we can either input 0 or 1 to the reversible classical query. It is the because of this classical limitation imposed by the classical laws of physics that does not allow us to compute the solution in one single query but requires two separate queries. This classical limitation is overcome in case of quantum solution. The input to the black-box/quantum query model of computation can be a superposition of 0 and 1 and thus allows simultaneous computation of both the inputs 0 and 1 to the function. Then using some manipulations, upon measurement we can get a distinctive results which can results in clearly distinguished two classical outcomes either 0 or 1 thus allowing us to identify whether the function  $f$  was constant or balanced. It is the power of superposition of states of 0 and 1 in application of the laws quantum mechanics and thus quantum computing that allows us to identify the function  $f$  in a single query.

### 7.3.2 Algorithm

**Claim:** If upon measurement of the first qubit, if it is  $|0\rangle$  then the function is constant or else the function is balanced.

**Analysis:** At inputs both the qubits  $q_0 = |0\rangle$  and  $q_1 = |0\rangle$ . As  $q_0$  passes through the Hadamard Gate it becomes  $|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$ . As the qubit  $q_1$  passes through X gate, its output is  $|1\rangle$  and then it is passed through H gate thus, its state at the output of the Hadamard gate will be  $|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$ . Thus, at the oracle we have  $|+\rangle$  and  $|-\rangle$  as inputs. Thus, the action of oracle is:

$$U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$$

Thus, the action of oracle will be as follows:

$$\begin{aligned} U_f |+\rangle |-\rangle &= U_f \left( \frac{|0\rangle + |1\rangle}{\sqrt{2}} \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \\ U_f \frac{1}{2} (|00\rangle - |01\rangle + |10\rangle - |11\rangle) &= \frac{1}{2} [|0\rangle |0 \oplus f(0)\rangle - |0\rangle |1 \oplus f(0)\rangle + |1\rangle |0 \oplus f(1)\rangle - |1\rangle |1 \oplus f(1)\rangle] \end{aligned}$$

Thus, the output of the oracle is as shown above. Recall that the output from the oracle was

$$\begin{aligned} &\frac{1}{2} [|0\rangle |0 \oplus f(0)\rangle - |0\rangle |1 \oplus f(0)\rangle + |1\rangle |0 \oplus f(1)\rangle - |1\rangle |1 \oplus f(1)\rangle] \\ &= \frac{1}{2} [|0\rangle (|0 \oplus f(0)\rangle - |1 \oplus f(0)\rangle) + |1\rangle (|0 \oplus f(1)\rangle - |1 \oplus f(1)\rangle)] \end{aligned}$$

Now using the fact that

$$|0 \oplus a\rangle - |1 \oplus a\rangle = (-1)^a (|0\rangle - |1\rangle)$$

we can simplify the above equation to

$$\frac{1}{2} [|0\rangle (-1)^{f(0)} (|0\rangle - |1\rangle) + |1\rangle (-1)^{f(1)} (|0\rangle - |1\rangle)]$$

which can be further simplified as

$$\frac{1}{2} [((-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle) (|0\rangle - |1\rangle)]$$

Now factorizing the qubits we get

$$\left( \frac{1}{\sqrt{2}} ((-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle) \right) \otimes \left( \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \right)$$

$$\begin{aligned}
&= \left( \frac{1}{\sqrt{2}} ((-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle) \right) \otimes |-\rangle \\
&= (-1)^{f(0)} \left( \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} (-1)^{f(0)\oplus f(1)} |1\rangle \right) \otimes |-\rangle
\end{aligned}$$

Note that  $(-1)^{f(0)}$  does not belong to a specific qubit of the two. When we applied the oracle, the second qubit was in the state  $|-\rangle$ . This is called querying in the phase. Also, note that  $(-1)^{f(0)}$  seems to be a global phase but it is not the global phase since the first qubit is in the superposition and depends on the value of  $f(0)$ . It's only a global phase if the first qubit is in a computational basis state, of the form  $|0\rangle$  or  $|1\rangle$ . It's not a global phase if the qubit is in superposition and changes depending on the value of  $f(0)$ . Now we apply the hadamard gate on the first qubit as shown in the circuit 7.3. We get

$$\begin{aligned}
&(H \otimes I) \left( \left( \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} (-1)^{f(0)\oplus f(1)} |1\rangle \right) \otimes (-1)^{f(0)} |-\rangle \right) \\
&= |f(0) \oplus f(1)\rangle \otimes (-1)^{f(0)} |-\rangle
\end{aligned}$$

Here, we have used the observation that

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

for  $a \in \{0, 1\}$ , which is again easily verified by considering the cases  $a = 0$  and  $a = 1$ . Thus, upon measurement of the first qubit, the measurement therefore results in the value  $f(0) \oplus f(1)$  which is one of the basis states either 0 or 1 with certainty. This value is 0 if  $f$  is constant and 1 if  $f$  is balanced. For further explanation consider the two cases:

### 1. Case 1: Function is Constant $f(0) = f(1) = 0$

If the function is constant then  $f(0) = f(1)$ . Let  $f(0) = f(1) = 0$  and thus the output of the oracle will be:

$$\frac{1}{2} [|0\rangle |0\rangle - |0\rangle |1\rangle + |1\rangle |0\rangle - |1\rangle |1\rangle]$$

Now factorizing the above into tensor product of two states as:

$$\begin{aligned}
\frac{1}{2} [|0\rangle |0\rangle - |0\rangle |1\rangle + |1\rangle |0\rangle - |1\rangle |1\rangle] &= \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \\
(H \otimes I)(|+\rangle \otimes |-\rangle) &= |0\rangle \otimes |-
\end{aligned}$$

Hence, as the qubit  $q_0$  passes through the Hadamard gate, the action of the Hadamard gate on the first qubit will be  $H|+\rangle = |0\rangle$ . Thus, upon measurement the output will be  $|0\rangle$  with probability 1. Thus, we get a measurement of 0.

**2. Case 1: Function is Constant  $f(0) = f(1) = 1$**

If the function is constant then  $f(0) = f(1)$ . Let  $f(0) = f(1) = 1$  and thus the output of the oracle will be:

$$\frac{1}{2} [|0\rangle |1\rangle - |0\rangle |0\rangle + |1\rangle |1\rangle - |1\rangle |0\rangle]$$

Now factorizing the above into tensor product of two states as:

$$\begin{aligned} \frac{1}{2} [|0\rangle |1\rangle - |0\rangle |0\rangle + |1\rangle |1\rangle - |1\rangle |0\rangle] &= \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes -\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\ (H \otimes I)(|+\rangle \otimes -|-\rangle) &= |0\rangle \otimes -|-\rangle \end{aligned}$$

Hence, as the qubit  $q_0$  passes through the Hadamard gate, the action of the Hadamard gate on the first qubit will be  $H|+\rangle = |0\rangle$ . Thus, upon measurement the output will be  $|0\rangle$  with probability 1. Thus, we get a measurement of 0.

**3. Case 2: Function is Balanced with  $f(0) = 0$  and  $f(1) = 1$**

If the function is balanced then  $f(0) \neq f(1)$  and let  $f(0) = 0$  and  $f(1) = 1$ . Thus, the output of the oracle will be:

$$\frac{1}{2} [|0\rangle |0\rangle - |0\rangle |1\rangle + |1\rangle |1\rangle - |1\rangle |0\rangle]$$

Now factorizing the above into tensor product of two states as:

$$\begin{aligned} \frac{1}{2} [|0\rangle |0\rangle - |0\rangle |1\rangle + |1\rangle |1\rangle - |1\rangle |0\rangle] &= \frac{|0\rangle - |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \\ (H \otimes I)(|-\rangle \otimes |-\rangle) &= |1\rangle \otimes |-\rangle \end{aligned}$$

Hence, as the qubit  $q_0$  passes through the Hadamard gate, the action of the Hadamard gate on the first qubit will be  $H|-\rangle = |1\rangle$ . Thus, upon measurement the output will be  $|1\rangle$  with probability 1. Thus, we get a measurement of 1.

**4. Case 2: Function is Balanced with  $f(0) = 1$  and  $f(1) = 0$**  If the function is balanced then  $f(0) \neq f(1)$  and let  $f(0) = 1$  and  $f(1) = 0$ . Thus, the output of the oracle will be:

$$\frac{1}{2} [|0\rangle |1\rangle - |0\rangle |0\rangle + |1\rangle |0\rangle - |1\rangle |1\rangle]$$

Now factorizing the above into tensor product of two states as:

$$\frac{1}{2} [|0\rangle |1\rangle - |0\rangle |0\rangle + |1\rangle |0\rangle - |1\rangle |1\rangle] = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \otimes -\frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

$$(H \otimes I)(|-\rangle \otimes -|-\rangle) = |1\rangle \otimes -|-\rangle$$

Hence, as the qubit  $q_0$  passes through the Hadamard gate, the action of the Hadamard gate on the first qubit will be  $H|-\rangle = |1\rangle$ . Thus, upon measurement the output will be  $|1\rangle$  with probability 1. Thus, we get a measurement of 1.

It is important to see that the first hadamard gate when applied to  $|0\rangle$  state produces the superposition state  $|0\rangle + |1\rangle$ . The hadamard applied to  $|1\rangle$  state results in  $|-\rangle$  state which helps induce phases to the output from the oracle thus changing it from  $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$  to  $\frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle + (-1)^{f(1)})|1\rangle$ . Then the Hadarmard applied to the first qubit extracts the phase difference and results in  $|f(0) \oplus f(1)\rangle$ . Thus, upon measurement, in case if the function is constant we get a measurement of 0 and if the function is balanced we get a measurement of 1. Thus, we can determine whether the function is constant or balanced using only one function call. This is a speedup over the classical algorithm which requires two function calls to determine whether the function is constant or balanced.

Notice what this algorithm does not do. It does not somehow extract both values of the function,  $f(0)$  and  $f(1)$ , with one single query. IN fact, if you run this algorithm, then you get no information about the value of  $f(0)$  itself. Also, you get no information about the value of  $f(1)$  itself. You only get information about  $f(0) \oplus f(1)$ .

### 7.3.3 Example

**Example 7.3.1. (One-out-of-four search)** Let  $f : \{0, 1\}^2 \rightarrow \{0, 1\}$  with the special property that it attains the value 1 at exactly one of the four points in its domain. There are four possibilities for such a function and here are their truth tables.

Function	Input	Output	Function	Input	Output
$f_{00}$	00	1	$f_{01}$	00	0
	01	0		01	1
	10	0		10	0
	11	0		11	0

(a)  $f_{00}$

Function	Input	Output	Function	Input	Output
$f_{10}$	00	0	$f_{11}$	00	0
	01	0		01	0
	10	1		10	0
	11	0		11	1

(b)  $f_{01}$

Function	Input	Output	Function	Input	Output
$f_{10}$	00	0	$f_{11}$	00	0
	01	0		01	0
	10	1		10	0
	11	0		11	1

(c)  $f_{10}$

(d)  $f_{11}$

Table 7.2: Possible Functions

The goal is to determine which one it is. In other words, given black-box computing such a function. Thus, access to the function is restricted to evaluations of the transformation  $B_f$ . We say that an evaluation of  $B_f$  on some input (quantum or classical) is a query. You're promised that it's one of these four, but you're not told which one. Your goal is to determine which of the four function it is. Thus, we can think of this problem as representing a simple search problem.

Classically, three queries are sufficient to solve the problem. You can query in the first three places and see if one of them evaluates to 1. If none of them do then, by process of elimination, you can deduce that the 1 must be the fourth place.

In the quantum case, one query will be sufficient to solve the problem. Here the oracle will perform operations implementing the function mapping two bits to one bit as

$$U_f |x\rangle |y\rangle |z\rangle = |x\rangle |y\rangle |z \oplus f(x, y)\rangle$$

There are two qubits for the input, and a third qubit for the output of the function. In the computational basis, the value of  $f(x, y)$  is XORed onto the third qubit. Of course, that description is for states in the computational basis states.

Similar to what we did in case of Deutsch's algorithm: by setting the target qubit to state  $|-\rangle$  so as to query in the phase, and then querying inputs in a uniform superposition. Consider the circuit as shown in the figure 7.4. The state after the Hadamard transform and just before the query can be written as

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

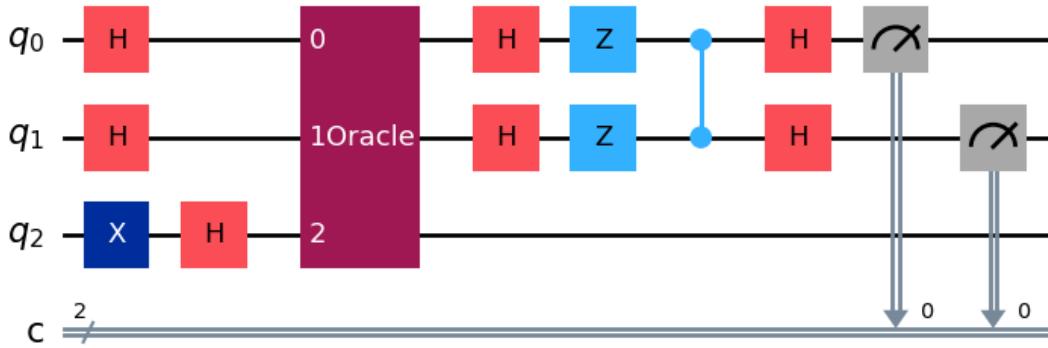


Figure 7.4: Quantum Circuit

After the transformation  $U_f$  is performed the state will be

$$\left( \frac{1}{2}(-1)^{f_{00}} |00\rangle + \frac{1}{2}(-1)^{f(01)} |01\rangle + \frac{1}{2}(-1)^{f_{10}} |10\rangle + \frac{1}{2}(-1)^{f(11)} |11\rangle \right) \otimes \left( \frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle \right)$$

The reasoning is same as in the analysis off the Deutsch's algorithm - we are using the **phase-kickback effect**.

Now we are left with four possibilities for the state of the first two qubits:

$$\begin{aligned} f = f_{00} &\implies |\phi_{00}\rangle = -\frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \\ f = f_{01} &\implies |\phi_{01}\rangle = \frac{1}{2}|00\rangle - \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \\ f = f_{10} &\implies |\phi_{10}\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle - \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \\ f = f_{11} &\implies |\phi_{11}\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle - \frac{1}{2}|11\rangle \end{aligned}$$

Note that these four states form an orthonormal set. This means that they are unit vectors and pairwise orthogonal.

$$\langle \phi_{ab} | \phi_{cd} \rangle = \begin{cases} 1 & \text{if } a = c \text{ and } b = d \\ 0 & \text{otherwise} \end{cases}$$

Whenever you have an orthonormal set like this, it is always possible to build a quantum circuit that exactly distinguishes the states. In fact, the corresponding

unitary transformation is easy to obtain: you let the vectors form the columns of a matrix and then take the conjugate transform. In this case we want this matrix:

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

You can check that  $U |\phi_{ab}\rangle = |ab\rangle$  for all possible choice of  $a, b \in \{0, 1\}$ . We can use the following gates that we have already seen to perform the unitary transformation  $U$  as seen in figure 7.4. Note that we have used a phase flip or  $\sigma_z$  gate and and Controlled -Z gate with the control being on the first qubit and the second qubit being the target bit. Recall that the phase flip gate or  $\sigma_z$  gate is

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

**Example 7.3.2.** Implement all the four functions in oracles and simulate the results on a quantum computer.

## 7.4 Implementation using Qiskit

### 7.4.1 Background

Usage estimate: 4 seconds (NOTE; This is an estimate only. Your runtime may vary)

In order to run Deutsch's algorithm requires an oracle that implements the function either constant or balanced and Deutsch's Circuit.

Here, we demonstrate how to construct Deutsch's oracles and use the gates from the Qiskit circuit library to easily set up a Deutsch's Circuit. The runtime *Sampler* primitive allows seamless execution of Deutsch's circuit.

### 7.4.2 Requirements

Before starting this tutorial, ensure that you have the following installed:

- Qiskit SDK 1.0 or later, with visualization support ( pip install 'qiskit[visualization]' )
- Qiskit Runtime (pip install qiskit-ibm-runtime) 0.22 or later

### 7.4.3 Setup

Here we import the small number of tools we need for this tutorial.

```

1 #Built-in modules
2 import math
3
4 #imports from Qiskit
5 from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
6 from qiskit.visualization import plot_distribution
7
8 #Imports from Qiskit Runtime
9 from qiskit_ibm_runtime import QiskitRuntimeService
10 from qiskit_ibm_runtime import SamplerV2 as Sampler

1 # To run on hardware, select the backend with the fewest number of jobs
    in the queue
2 service = QiskitRuntimeService(channel="ibm_quantum")
3 backend = service.least_busy(operational=True, simulator=False)
4 backend.name

```

This output one of the ibm quantum computers (say 'ibm\_brisbane').

#### 1. Step 1: Map classical inputs to a quantum problem:

There are four possibilities for the oracles for the case of Deutsch Circuit where the oracle implements the function  $f : \{0, 1\} \rightarrow \{0, 1\}$  which could either be constant ( $f_1$  or  $f_2$ ) or constant ( $f_3$  or  $f_4$ ) as given in table 7.1. The code used for creating the Deutsch Circuit is as follows:

```

1 def deutsch_circuit(value):
2     # Create a Quantum Circuit
3     qubits=QuantumRegister(2,name="q")
4     cbits=ClassicalRegister(1,name="c")
5     circuit=QuantumCircuit(qubits, cbits)
6
7     q0,q1=qubits
8     circuit.h(q0)
9     circuit.x(q1)
10    circuit.h(q1)
11
12    circuit.barrier()
13    if(value==1):
14        circuit.id(q0)
15        circuit.id(q1)
16    elif(value==2):
17        circuit.id(q0)
18        circuit.x(q1)

```

```

19     elif(value==3):
20         circuit.cx(q0,q1)
21     elif(value==4):
22         circuit.cx(q0,q1)
23         circuit.x(q1)
24
25     circuit.barrier()
26     circuit.h(q0)
27
28     circuit.measure(q0,0)
29
30     return circuit
31

```

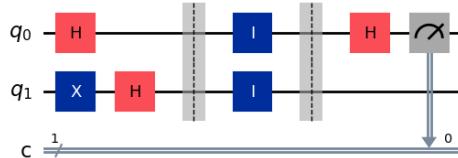
For creating one of the circuits, use the following code:

```

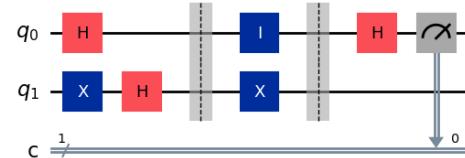
1 qc_4=deutsch_circuit(value=4)
2 qc_4.draw("mpl")
3

```

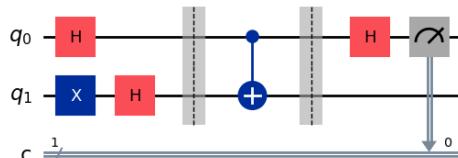
All the circuits along with their corresponding oracles implementing the four functions are as shown in the figure below:



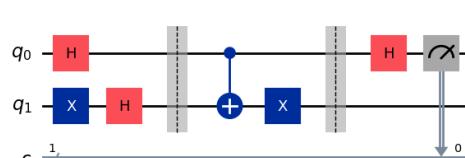
(a)  $f = f_1$



(b)  $f = f_2$



(c)  $f = f_3$



(d)  $f = f_4$

Figure 7.5: Deutsch Circuit implementation for possible functions along with their corresponding oracles

Note that each of the oracle (shown between the two barriers) implements the required function. Thus the above shown figure corresponds to the Deutsch circuit with its functions in the oracle implementation.

## 2. Step 2: Optimize Problems for Quantum Execution

```
1 from qiskit.transpiler.preset_passmanagers import
2
3 generate_preset_pass_manager
4
5 target = backend.target
6 pm = generate_preset_pass_manager(target=target,
7                                     optimization_level=3)
8
9 circuit_is_a_1 = pm.run(qc)
9 circuit_is_a_1.draw(output="mpl", idle_wires=False, style="iqp")
```

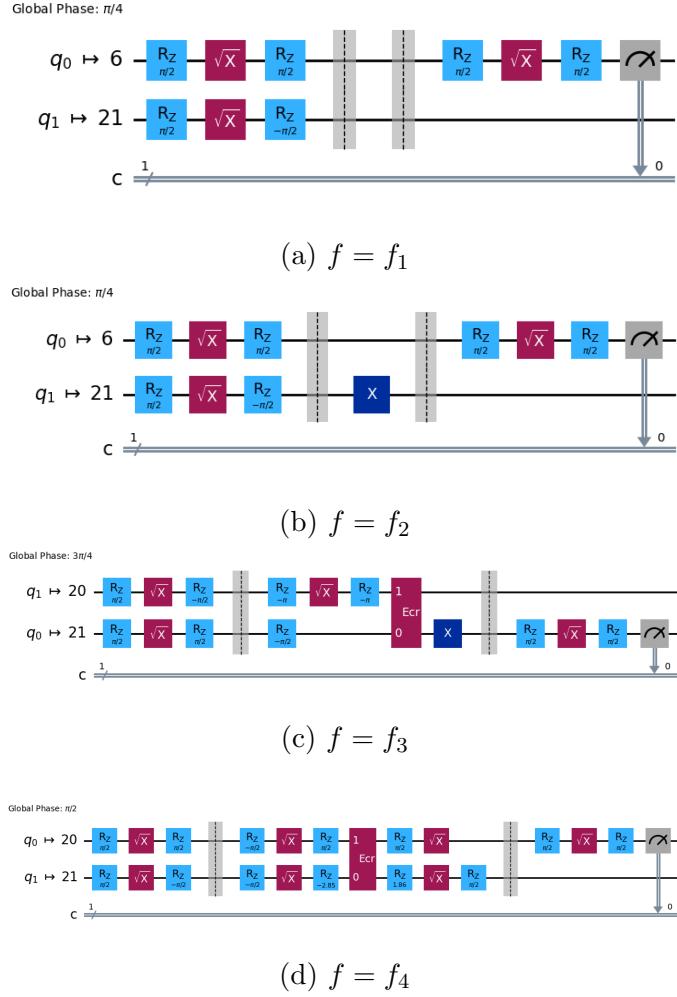


Figure 7.6: Corresponding Optimized Deutsch Circuit

### 3. Step 3: Execute using Qiskit Primitives

We are now required to run the circuit and sample the measurements on the first qubit. Thus for execution we will use the Sampler runtime primitive.

```

1 # TO run on local simulator:
2 #1. Use the StatevectorSimulator from qiskit.primitives instead
3 sampler=Sampler(backend=backend)
4 sampler.options.default_shots=10_000
5 result_1 = sampler.run([circuit_is_a_1]).result

```

```
6 dist_1 = result_1[0].data.c.get_counts()  
7
```

The output of the above code will be the distribution of the measurements on the first qubit. Similar code can be written for all the four possible functions.

#### 4. Step 4: Post-Process Results

Now we plot a histogram of the measurements on the first qubit in the classical format.

```
1 from qiskit.visualization import plot_histogram  
2 plot_histogram(dist_1)  
3
```

The output of the above code will be the histogram of the distribution of the measurements on the first qubit. Similar code can be written for all the four possible functions.

The following output is produced upon running this code on ibm\_brisbane as shown in figure 7.7. The histogram plot is done for 10000 shots and results can be clearly seen the results as expected. The minor deviations in the results is due to the inherent errors (because of the noise) in the quantum computer and its probabilistic nature in the measurement process.

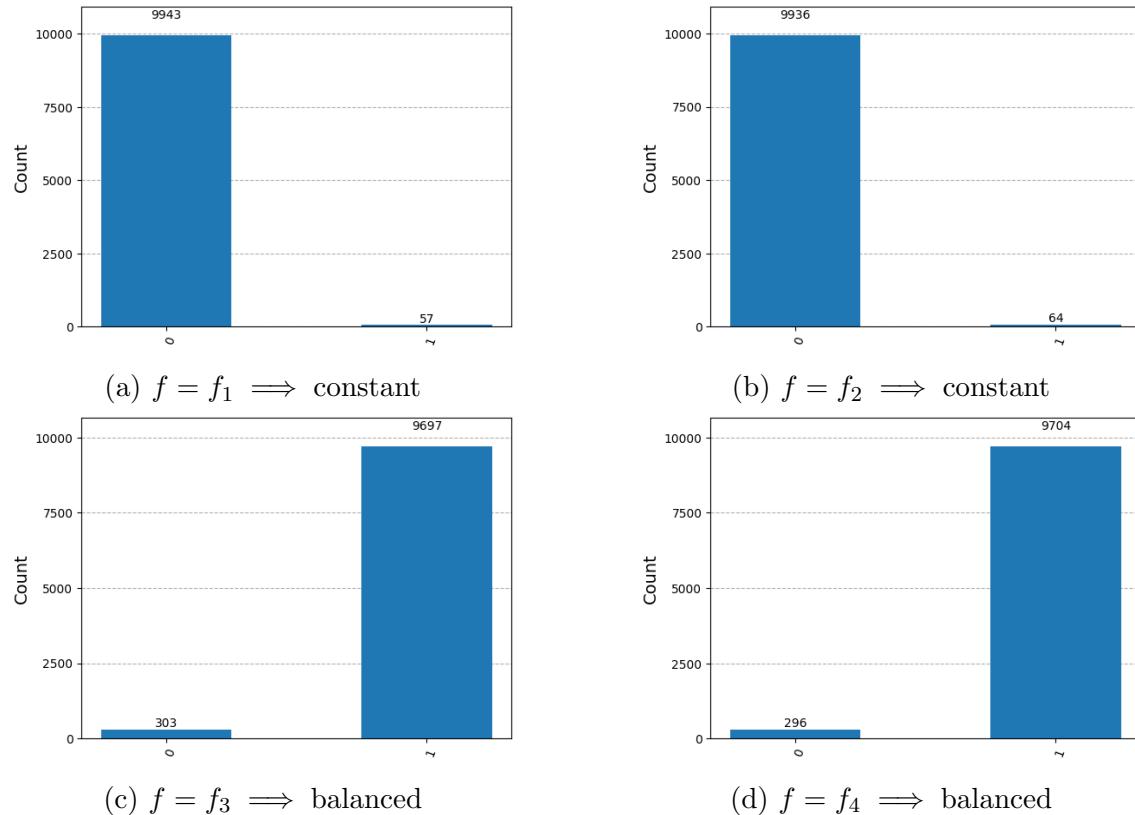


Figure 7.7: Results

We can see in the figure 7.7a that the result of the measurement of the first qubit is 0 (9943 times) and 1 (57 times). Thus, we can conclusively say that the function implemented was a constant function (inside the oracle) which we know is true. Hence, the results are expected. Similar conclusions can be made for all the other results and we can clearly see that they agree with what we implemented and expected. Here, we are able to tell whether the function was constant or balanced but not what the function was out of the four different possibilities. Thus the results are in agreement with the theory showing that the function implemented in the oracle can be identified if it is constant or balanced based on whether the output is 0 or 1 respectively. Note that  $f = f_1$  was a constant function and was accordingly implemented inside the oracle, now looking at the results we can conclusively say that the function was a constant function. Similar can be said for all the four functions implemented.

Note that the results might differ for different runs because of the noise in quantum computer.

# Chapter 8

## Deutsch-Jozsa Algorithm

This is a generalization of the Deutsch Algorithm which generalizes the functions from  $n$  bits to 1 bit. This quantum algorithm solves the problem with exponentially fewer queries than any classical algorithm.

### 8.1 The Problem Definition

**Definition 8.1.1.** Given a classical function  $f : \{0,1\}^n \rightarrow \{0,1\}$ , which takes  $n$  input bits and returns a single output bit. The function is called constant if  $f(x) = 0$  or  $f(x) = 1$  for all  $x \in \{0,1\}^n$  and the function is called balanced if  $f(x) = 0$  for exactly half of the inputs.

Our aim is to find whether the function  $f$  is constant or balanced with as minimum number of queries as possible to the Oracle.

The total number of such possible functions are  $2^{2^n}$  out of which 2 are constant and  ${}^{2^n}C_{2^{n-1}} = \frac{2^n!}{2^{n-1}!(2^n - 2^{n-1})!} \approx \frac{2^n}{\sqrt{n}}$  are balanced functions for  $n$  input bits and 1 output bit. The rest are neither. Say for the case  $n=2$ , the possible functions are as shown in the table 8.1. We are given a black-box computing a function that is promised to be either constant or balanced, but we are not told which one. Our goal is to figure out which of the two cases it is, with as few queries to  $f$  as possible.

Function	Input	Output
$f_1$	00	0
	01	0
	10	0
	11	0
$f_2$	00	1
	01	1
	10	1
	11	1
$f_3$	00	0
	01	0
	10	1
	11	1
$f_4$	00	0
	01	1
	10	1
	11	0
$\vdots$		$\vdots$
$f_9$	00	0
	01	1
	10	1
	11	0

(a) Constant Function

(b) Balanced Function

Table 8.1: Possible Functions

## 8.2 The Classical Solution

**On a Classical Computer:** Note that even if you query the function in many spots and always see the same value, you still might not know which way it goes. It could be constant. But it could also be that, in many of the places that you did not query, the function takes the opposite value and it's actually a balanced function. It's only after you've queried in more than half of the spots that you can be sure about which case it is. Therefore, for number of queries that a classical algorithm must make is  $2^{n+1} + 1$ . In order to determine whether the function is constant or balanced we need to query the function for just 1 more than half of the possible inputs. Thus, we require  $2^{n-1} + 1$  queries to determine whether the function is constant or balanced. Thus, the time complexity for this on a classical computer is  $\mathcal{O}(2^n)$  i.e. exponential time complexity. That's a lot of queries when  $n$  is large. If all the inputs produce the same output then the function is constant else the function is balanced.

## 8.3 The Quantum Solution: Deutsch-Jozsa Algorithm

**On a Quantum Computer:** To solve this problem we need to query the function only once on a Quantum Computer using the Deutsch-jozsa Algorithm. Thus, the time complexity to solve this problem is  $\mathcal{O}(1)$  i.e. constant time complexity. This is an exponential speedup over the classical algorithm.

### 8.3.1 Idea of the Algorithm

Similar to the case of Deutsch algorithm, we start off as usual, setting the target qubit to state  $|-\rangle$  and seeing the other qubits - those where the inputs to  $f$  are - into a uniform superposition of all  $n$ -qubit state. That's what the Hadamard to each of  $n$  qubits in state  $|0\rangle$  does as shown in figure 8.1. Thus resulting phase kickback (also called querying in the phase) on all the qubits except the target bit. Then applying a hadamard transform on all the  $n$  qubits to extract the phase difference. Finally, performing a measurement which can clearly distinguish between constant, if the measurement reveals all 0 and balanced, if the measurement reveals all 1 in the classical state, Thus, determining functions implemented inside the oracle.

### 8.3.2 Algorithm

The circuit for Deutsch-Jozsa Algorithm Circuit is as shown in the figure 8.1.

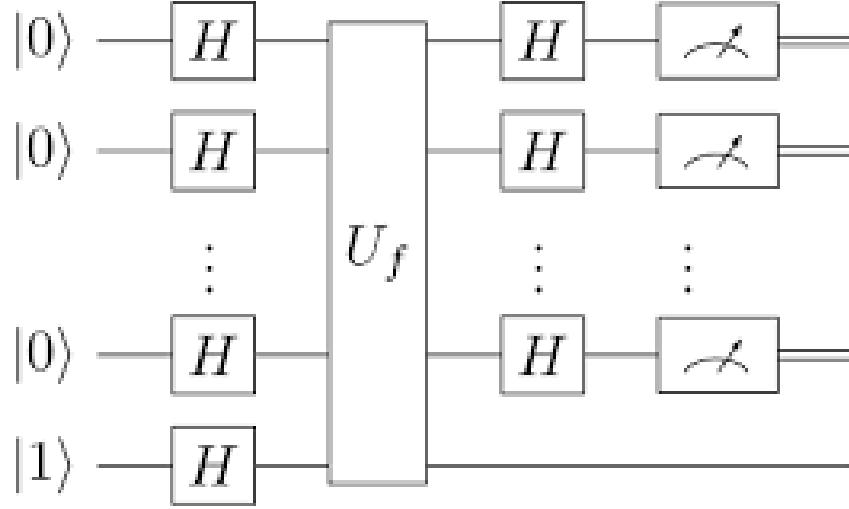


Figure 8.1: Deutsch-Jozsa Circuit

The input of this circuit is  $q_0 = |0\rangle, q_1 = |0\rangle, \dots, q_{n-1} = |0\rangle, q_n = |1\rangle$ . Now at the input we have  $|0\rangle^{\otimes n}|1\rangle$  and the action of the Hadamard gate on the input will be:

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

Now the oracle will show the action  $U_f$  as  $U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$  (Here  $x \in \{0,1\}^n$  and thus the action of the oracle will be:

$$\begin{aligned} U_f \left( \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \frac{(|0\rangle - |1\rangle)}{\sqrt{2}} \right) &= U_f \left( \frac{1}{\sqrt{2^n}} \frac{1}{\sqrt{2}} \sum_{x \in \{0,1\}^n} |x\rangle|0\rangle - |x\rangle|1\rangle \right) \\ &= \frac{1}{\sqrt{2^n}} \frac{1}{\sqrt{2}} \sum_{x \in \{0,1\}^n} U_f|x\rangle|0\rangle - U_f|x\rangle|1\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \left( \frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} \right) \end{aligned}$$

Now  $|0 \oplus f(x)\rangle = |f(x)\rangle$  and  $|1 \oplus f(x)\rangle = |\overline{f(x)}\rangle$  where  $\overline{f(x)}$  is the complement of  $f(x)$ . Thus, substituting this in the equation we get,

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \left( \frac{|f(x)\rangle - |\overline{f(x)}\rangle}{\sqrt{2}} \right)$$

Now when  $f(x) = 0$  then  $\frac{|f(x)-|\overline{f(x)}\rangle\rangle}{\sqrt{2}} = \frac{|0\rangle-|1\rangle}{\sqrt{2}} = |- \rangle$  and when  $f(x) = 1$  then  $\frac{|f(x)-|\overline{f(x)}\rangle\rangle}{\sqrt{2}} = \frac{|1\rangle-|0\rangle}{\sqrt{2}} = -|- \rangle$ . Thus, this can be written as:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \frac{(|f(x)\rangle - |\overline{f(x)}\rangle)}{\sqrt{2}} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle |- \rangle$$

Note that the first n qubits form a uniform superposition of all  $2^n$  computational basis states of n qubits, with a phase of  $(-1)^{f(x)}$  for each  $|x\rangle$ . IN constant case, either all the phases are +1 or all the phases are -1. So the state remains in a uniform superposition of computational basis states and just picks up a global phase of +1 or -1. In balanced case, there are lots of possibilities, depending on which particular balanced function it is. But one thing we can say is that the state is orthogonal to the state that arises in the constant case. Because, in the computational basis, the state will have +1 in half of its components and -1 in the other half. So when you take the dot product, with a vector that has (say) +1 in each component you get an equal number of +1s and -1s, which cancel and results in zero. Note that since the cases that we are trying to distinguish between result in orthogonal state, in principle, they are perfectly distinguishable.

Now we apply the Hadamard gate on the first n qubits and the action of the Hadamard gate on the first n qubits will be:

$$\begin{aligned} (H^{\otimes n} \otimes I) \left( \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle |- \rangle \right) &= H^{\otimes n} \left( \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle \right) \otimes I |- \rangle \\ &= \left( \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} H^{\otimes n} |x\rangle \right) \otimes |- \rangle \end{aligned}$$

Recall that  $H^{\otimes n} |x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle$  for Hadamard gate (refer section 3.2.2) acting on n qubits. Thus, substituting this in the above equation can be written as:

$$\left( \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \sum_{y \in \{0,1\}^n} (-1)^{f(x)+x \cdot y} |y\rangle \right) \otimes |- \rangle$$

Now, we perform the measurement on the first n qubits. In the constant case, this transforms the state of the  $\pm |00\dots 0\rangle = \pm |0^n\rangle$ . IN the balanced case, since unitary operations preserve orthogonality relationships, the state is transformed to some

state that is orthogonal to  $|0^n\rangle$ . Now, all we really care about is the probability that the measurements all give outcome 0. The amplitude associated with the classical state  $|0^n\rangle$  is

$$\frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x)}$$

Thus, the probability that the measurement all give outcome 0 is

$$\left| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} \right|^2 = \begin{cases} 1 & \text{if } f \text{ is constant} \\ 0 & \text{if } f \text{ is balanced} \end{cases}$$

Consider the following two cases:

### 1. Case 1: Function is Constant

- Let  $f(x) = 0$  for all  $x \in \{0,1\}^n$ . Hence we need to perform measurement on the first n qubits of the following state:

$$\left( \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle \right) \otimes |- \rangle$$

Upon simplifying we get,

$$\left( \frac{1}{2^n} 2^n |0^{\otimes n}\rangle \right) \otimes |- \rangle = |0^{\otimes n}\rangle \otimes |- \rangle$$

Thus, upon measurement the output will be  $|0^{\otimes n}\rangle$  with probability 1. Thus, we get a measurement of  $|0^{\otimes n}\rangle$ .

- Let  $f(x) = 1$  for all  $x \in \{0,1\}^n$ . Hence we need to perform measurement on the first n qubits of the following state:

$$\left( \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \sum_{y \in \{0,1\}^n} (-1)^{1+x \cdot y} |y\rangle \right) \otimes |- \rangle$$

Upon simplifying we get,

$$\left( \frac{1}{2^n} 2^n |0^{\otimes n}\rangle \right) \otimes |- \rangle = -|0^{\otimes n}\rangle \otimes |- \rangle$$

Thus, upon measurement the output will be  $|0^{\otimes n}\rangle$  with probability 1. Thus, we get a measurement of  $|0^{\otimes n}\rangle$ .

## 2. Case 2: Function is Balanced

- Let  $f(x)$  be a balanced function. Thus,  $f(x) = 0$  for exactly half of the inputs and  $f(x) = 1$  for the other half of the inputs. Hence we need to perform measurement on the first  $n$  qubits of the following state:

$$\left( \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \sum_{y \in \{0,1\}^n} (-1)^{f(x)+x \cdot y} |y\rangle \right) \otimes |- \rangle$$

The output of this depends on the function but it will surely not be  $|0^n\rangle$ .

If the outcome is  $00\dots 0 = 0^n$  then we report "constant" and if the outcome is any string that's not  $0^n$  (not all zeroes) then we report "balanced". Note that in the balanced case, it is impossible to get outcome  $0^n$ , because measuring a state orthogonal to  $0^n$  cannot result in outcome  $0^n$ .

Thus, the one-query algorithm for the constant vs balanced problem works. It achieves something with one single query that requires exponentially many queries by any classical algorithm.

### 8.3.3 Example

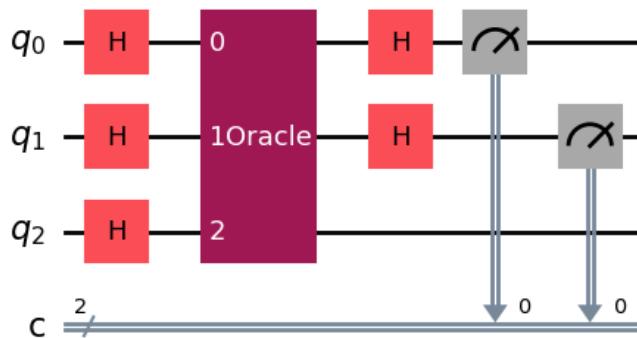


Figure 8.2: Deutsch-jozsa Algorithm for  $n=2$

**Example 8.3.1.** Consider the Deutsch-jozsa algorithm for  $n=2$ . The possible functions are as shown in the table 8.1 and the circuit for the Deutsch-jozsa algorithm is as shown in the figure 8.2. The input to the circuit is  $|00\rangle|1\rangle$ . The action of the Hadamard gate on the input will be:

$$\begin{aligned} H^{\otimes 2}|00\rangle H|1\rangle &= \frac{1}{2} \sum_{x \in \{0,1\}^2} |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \\ &= \frac{1}{2\sqrt{2}} (|000\rangle + |010\rangle + |100\rangle + |110\rangle - |001\rangle - |011\rangle - |101\rangle - |111\rangle) \end{aligned}$$

The action of the oracle will be:

$$\begin{aligned} U_f &\left( \frac{1}{2\sqrt{2}} (|000\rangle + |010\rangle + |100\rangle + |110\rangle - |001\rangle - |011\rangle - |101\rangle - |111\rangle) \right) \\ &= \frac{1}{2\sqrt{2}} (U_f|000\rangle + U_f|010\rangle + U_f|100\rangle + U_f|110\rangle - U_f|001\rangle - U_f|011\rangle - U_f|101\rangle - U_f|111\rangle) \\ &= \frac{1}{2\sqrt{2}} (|00\rangle|f(00)\rangle + |01\rangle|f(01)\rangle + |10\rangle|f(10)\rangle + |11\rangle|f(11)\rangle - |00\rangle|\overline{f(00)}\rangle \\ &\quad - |01\rangle|\overline{f(01)}\rangle - |10\rangle|\overline{f(10)}\rangle - |11\rangle|\overline{f(11)}\rangle) \end{aligned}$$

Now consider the following cases:

### 1. Case 1: $f(x)$ is constant

- Let  $f(x) = 0$  for all  $x \in \{0,1\}^2$ . Thus, the output of the oracle will be:

$$\begin{aligned} &\frac{1}{2\sqrt{2}} (|00\rangle|0\rangle + |01\rangle|0\rangle + |10\rangle|0\rangle + |11\rangle|0\rangle - |00\rangle|1\rangle - |01\rangle|1\rangle - |10\rangle|1\rangle - |11\rangle|1\rangle) \\ &= \frac{1}{2\sqrt{2}} (|000\rangle + |010\rangle + |100\rangle + |110\rangle - |001\rangle - |011\rangle - |101\rangle - |111\rangle) \\ &= \frac{1}{2} (|00\rangle + |01\rangle + |10\rangle + |11\rangle) \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \end{aligned}$$

Now we apply Hadamard on the first  $n$  qubits and we get:

$$\begin{aligned} &(H^{\otimes 2} \otimes I) \frac{1}{2} ((|00\rangle + |01\rangle + |10\rangle + |11\rangle) \otimes |-\rangle) \\ &= \frac{1}{2} (H^{\otimes 2}|00\rangle + H^{\otimes 2}|01\rangle + H^{\otimes 2}|10\rangle + H^{\otimes 2}|11\rangle) \otimes |-\rangle \\ &= \frac{1}{2} \left[ \frac{1}{2} (|00\rangle + |01\rangle + |10\rangle + |11\rangle) + \frac{1}{2} (|00\rangle - |01\rangle + |10\rangle - |11\rangle) \right] \otimes |-\rangle \end{aligned}$$

Upon simplifying we get,

$$\frac{1}{4}(4|00\rangle) \otimes |-\rangle = |00\rangle \otimes |-\rangle$$

Thus, upon measurement the output will be  $|00\rangle$  with probability 1. Thus, we get a measurement of  $|00\rangle$ .

- Let  $f(x) = 1$  for all  $x \in \{0, 1\}^2$ . Thus, the output of the oracle will be:

$$\begin{aligned} & \frac{1}{2\sqrt{2}}(|00\rangle|1\rangle + |01\rangle|1\rangle + |10\rangle|1\rangle + |11\rangle|1\rangle - |00\rangle|0\rangle - |01\rangle|0\rangle - |10\rangle|0\rangle - |11\rangle|0\rangle) \\ &= \frac{-1}{2\sqrt{2}}(|000\rangle + |010\rangle + |100\rangle + |110\rangle - |001\rangle - |011\rangle - |101\rangle - |111\rangle) \\ &= \frac{-1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \end{aligned}$$

Now we apply Hadamard on the first n qubits and we get:

$$\begin{aligned} & (H^{\otimes 2} \otimes I) \frac{-1}{2}((|00\rangle + |01\rangle + |10\rangle + |11\rangle) \otimes |-\rangle) \\ &= \frac{-1}{2}(H^{\otimes 2}|00\rangle + H^{\otimes 2}|01\rangle + H^{\otimes 2}|10\rangle + H^{\otimes 2}|11\rangle) \otimes |-\rangle \\ &= \frac{-1}{2}[\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) + \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle) \\ &\quad - \frac{1}{2}(|00\rangle + |01\rangle - |10\rangle - |11\rangle) + \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle + |11\rangle)] \otimes |-\rangle \end{aligned}$$

Upon simplifying we get,

$$\frac{-1}{4}(4|00\rangle) \otimes |-\rangle = |00\rangle \otimes -|-\rangle$$

Thus, upon measurement the output will be  $|00\rangle$  with probability 1. Thus, we get a measurement of  $|00\rangle$ .

## 2. Case 2: $f(x)$ is balanced

- Let  $f(x)$  be a balanced function. Thus,  $f(x) = 0$  for exactly half of the inputs and  $f(x) = 1$  for the other half of the inputs. Then it can be shown that based upon the input chosen the output will be either of the first n qubits,  $|11\rangle$  or  $-|11\rangle$ . Thus, upon measurement the output will be  $|11\rangle$  with probability 1. Thus, we get a measurement of  $|11\rangle$ .

## 8.4 Complexity Analysis

Note that the classical query cost is expensive only if we require absolutely perfect performance. If a classical procedure queries  $f$  in random locations then, in case of a balanced function, it would have to be very unlucky to draw the same bit value.

Here's a classical probabilistic procedure that makes just two queries and performs fairly well. It selects two locations randomly (independently) and then outputs "constant" if the two bits are the same and "balanced" if the two bits values are different.

What happens if  $f$  is constant? In that case the algorithm always succeeds. What happens if  $f$  is balanced? In that case the algorithm succeeds with probability  $\frac{1}{2}$ . The probability that the two bits will be different will be  $\frac{1}{2}$ .

By repeating the above procedure  $k$  times, we can make the error probability exponentially small with respect to  $k$ . Only 4 queries are needed to obtain success probability  $3/4$ . And the success probability can be made to any constant, arbitrarily close to 1, with a constant number of queries.

In summary, we have considered three problems in the black-box model. For each problem, a quantum algorithms solves it with just one query, but more queries are required by classical Algorithm.

Problem	Quantum Solution	Classical Deterministic	Classical Probabilistic
Deutsch	1	2	2
Deutsch-Jozsa	1	$2^{n-1} + 1$	$\mathcal{O}(1)$

Table 8.2: Summary of Query costs for the problems

For Deutsch's problem, any classical algorithm requires 2 queries. And, for the Deutsch-Jozsa problem, any classical algorithm requires exponentially many queries to solve the problem perfectly. However, there is a probabilistic classical algorithm that makes only a constant number of queries and solves the problem with bounded error probability.

Along this line of thought, the following question seems natural. Is there a black-box problem for which the quantum-vs-classical query cost separation is stronger? For example, for which even probabilistic classical algorithms with bounded-error probability will require exponentially more queries than a quantum algorithm?

These problems seem rather unnatural, and it is difficult to imagine any realistic setting in which one would need or want to solve these problems. But perhaps more importantly, so far we have not demonstrated any significant advantage of quantum

algorithms over probabilistic algorithms. We'll address this question in the next chapter.

## 8.5 Conclusion

From the above algorithm, we can clearly see that for determining whether the function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  is constant or balanced we need to input the function with  $n |0\rangle$  and  $1 |1\rangle$  and query the function only once. Then, in the final measurement we measure the first  $n$  qubits and if the function is constant we get  $|0^{\otimes n}\rangle$  and if the function is balanced we get  $|11\rangle$ . Thus, we can determine whether the function is constant or balanced with just one query to the Oracle. This is an exponential speedup over the classical algorithm which requires  $2^{n-1} + 1$  queries to determine whether the function is constant or balanced. Thus, the time complexity of this algorithm is  $\mathcal{O}(1)$  i.e. constant time complexity. Hence, an exponential speedup over the classical algorithm.

## 8.6 Implementation using Qiskit

### 8.6.1 Background

Usage estimate: 4 seconds on ibm\_brisbane(NOTE; This is an estimate only. Your runtime may vary)

In order to run Deutsch-Jozsa algorithm requires an oracle that implements the function either constant or balanced and Deutsch-Jozsa Circuit.

Here, we demonstrate how to construct Deutsch-Josza oracles and use the gates from the Qiskit circuit library to easily set up a Deutsch-Jozsa Circuit.

### 8.6.2 Requirements

Before starting this tutorial, ensure that you have the following installed:

- Qiskit SDK 1.0 or later, with visualization support ( pip install 'qiskit[visualization]' )
- Qiskit Runtime (pip install qiskit-ibm-runtime) 0.22 or later

### 8.6.3 Setup

Here we import the small number of tools we need for this tutorial.

```

1 #Built-in modules
2 import math
3
4 #imports from Qiskit
5 from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
6 from qiskit.visualization import plot_distribution
7
8 #Imports from Qiskit Runtime
9 from qiskit_ibm_runtime import QiskitRuntimeService
10 from qiskit_ibm_runtime import SamplerV2 as Sampler

1 # To run on hardware, select the backend with the fewest number of jobs
    # in the queue
2 service = QiskitRuntimeService(channel="ibm_quantum")
3 backend = service.least_busy(operational=True, simulator=False)
4 backend.name

```

This output one of the ibm quantum computers (say 'ibm\_brisbane').

### 1. Step 1: Map classical inputs to a quantum problem:

Deutsch-Jozsa algorithm requires an oracle that implements the function either constant or balanced. Here, we demonstrate how to construct Deutsch-Jozsa oracles and use the gates from the Qiskit circuit library to easily set up a Deutsch-Jozsa Circuit. Here, we will work the with 2 qubits for constant and balanced functions and verify the results as expected.

The runtime Sampler primitive allows seamless execution of Deutsch circuit.

```

1 def deutsch_jozsa(num_qubits=2,value=0,oracle=True,oracle_type=""
2   constant",b_str=""):
3   # Create a Quantum Circuit
4   qubits=QuantumRegister(num_qubits+1,name="q")
5   cbits=ClassicalRegister(num_qubits,name="c")
6   circuit=QuantumCircuit(qubits, cbits)
7
8   circuit.x(qubits[-1])
9
10  for qubit in qubits:
11      circuit.h(qubit)
12
13  circuit.barrier()
14
15  if(oracle==True):
16      oracle_qc=QuantumRegister(num_qubits+1)
17      oracle=QuantumCircuit(oracle_qc,name="Oracle")
18      if(oracle_type=="constant"):

```

```

18     if (value==1):
19         oracle.x(oracle_qc[-1])
20         oracle.name="Const Oracle: 1"
21     elif (value==0):
22         oracle.id(oracle_qc[-1])
23         oracle.name="Const Oracle: 0"
24     else:
25         raise ValueError("Value must be 0 or 1")
26     elif (oracle_type=="balanced"):
27         if (b_str==""):
28             raise ValueError("Balanced Oracle requires a bit
29 string")
30         if (len(b_str)!=num_qubits):
31             raise ValueError("Bit string must be of length
32 equal to number of qubits")
33         for qubit in range(num_qubits):
34             if (b_str[qubit]=="1"):
35                 oracle.x(oracle_qc[qubit])
36
37         for qubit in oracle_qc[:-1]:
38             oracle.cx(qubit, oracle_qc[-1])
39
40         for qubit in range(num_qubits):
41             if (b_str[qubit]=="1"):
42                 oracle.x(oracle_qc[qubit])
43             oracle.name="Balanced Oracle"
44     else:
45         raise ValueError("Oracle type must be 'constant' or
46 'balanced'")
47
48     circuit.append(oracle, qubits)
49
50 else:
51     if (oracle_type=="constant"):
52         if (value==1):
53             circuit.x(qubits[-1])
54         elif (value==0):
55             circuit.id(qubits[-1])
56         else:
57             raise ValueError("Value must be 0 or 1")
58     elif (oracle_type=="balanced"):
59         if (b_str==""):
60             raise ValueError("Balanced Oracle requires a bit
61 string")
62         if (len(b_str)!=num_qubits):
63             raise ValueError("Bit string must be of length
64 ")

```

```

equal to number of qubits")
60     for qubit in range(num_qubits):
61         if(b_str[qubit]== "1"):
62             circuit.x(qubits[qubit])
63
64     for qubit in qubits[:-1]:
65         circuit.cx(qubit,qubits[-1])
66
67     for qubit in range(num_qubits):
68         if(b_str[qubit]== "1"):
69             circuit.x(qubits[qubit])
70     else:
71         raise ValueError("Oracle type must be 'constant' or "
72                         "balanced'")
72     circuit.barrier()
74
75     for qubit in qubits:
76         circuit.h(qubit)
77
78     for qubit in qubits[:-1]:
79         circuit.measure(qubit,qubits.index(qubit))
80
81     return circuit
82

```

Note the above code can create only certain type o balanced functions and not all its variants. For creating the function  $f = f_3$ , the following code is used.

```

1 # Create a Quantum Circuit
2 qubits=QuantumRegister(3,name="q")
3 cbits=ClassicalRegister(2,name="c")
4 circuit=QuantumCircuit(qubits,cbits)
5
6 q0,q1,q2=qubits
7 circuit.h(q0)
8 circuit.h(q1)
9 circuit.x(q2)
10 circuit.h(q2)
11
12 circuit.barrier()
13
14 circuit.cx(q0,q2)
15
16 circuit.barrier()
17

```

```

18 circuit.h(q0)
19 circuit.h(q1)
20 circuit.measure(q0,0)
21 circuit.measure(q1,1)
22
23 circuit.draw("mpl")
24

```

The rest of the code corresponding to  $f = f_3$  follows the similar pattern. For creating constant oracle circuit, use the following code:

```

1 qc_const_0=deutsch_jozsa(num_qubits=2,value=0,oracle=True,
                           oracle_type="balanced",b_str="00")
2 qc_const_0.draw("mpl")
3

```

For a balanced oracle, the code used is as follows:

```

1 qc_bal_0=deutsch_jozsa(num_qubits=2,value=1,oracle=False,
                         oracle_type="balanced",b_str="00")
2 qc_bal_0.draw("mpl")
3

```

Similarly any other circuit can be created by changing the input parameters. The circuit along with the corresponding oracle implementing the function is as shown in the figure 8.3 according to the table 8.1.

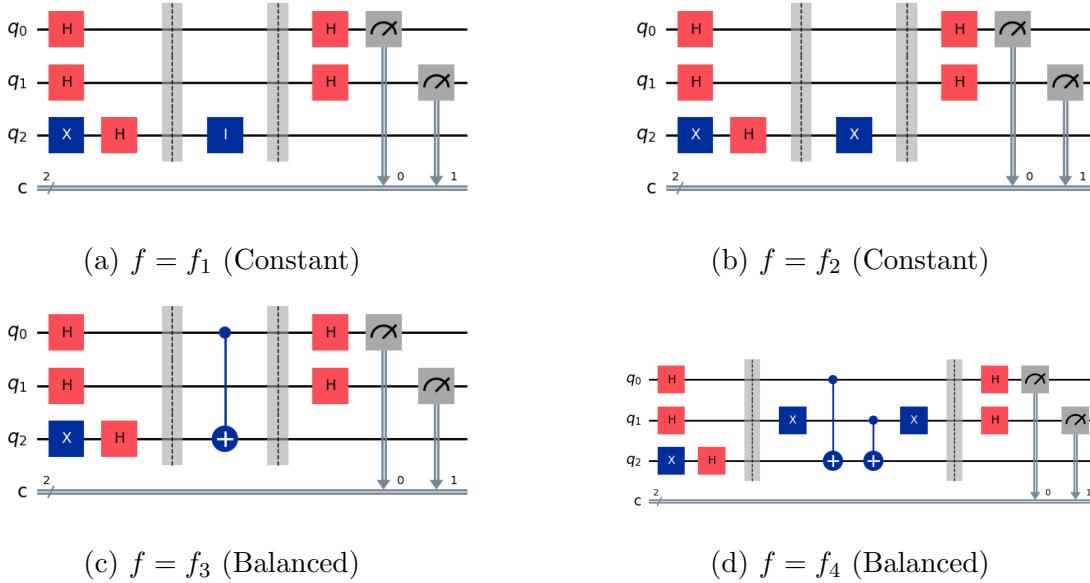


Figure 8.3: Deutsch-Jozsa Circuit implementation for few of the possible functions along with their corresponding oracles

Note that the oracle (shown between the two barriers) implements the required functions. Thus the above shown figure corresponds to the Deutsch-Jozsa circuit with a few of the possible function.

## 2. Step 2: Optimize Problems for Quantum Execution

```

1 from qiskit.transpiler.preset_passmanagers import
   generate_preset_pass_manager
2
3 target=backend.target
4 pm = generate_preset_pass_manager(backend=backend,
   optimization_level=3)
5
6 circuit_is_a_const_0=pm.run(qc_const_0)
7 circuit_is_a_const_0.draw(output="mpl", idle_wires=False, style="iqp")
  )

```

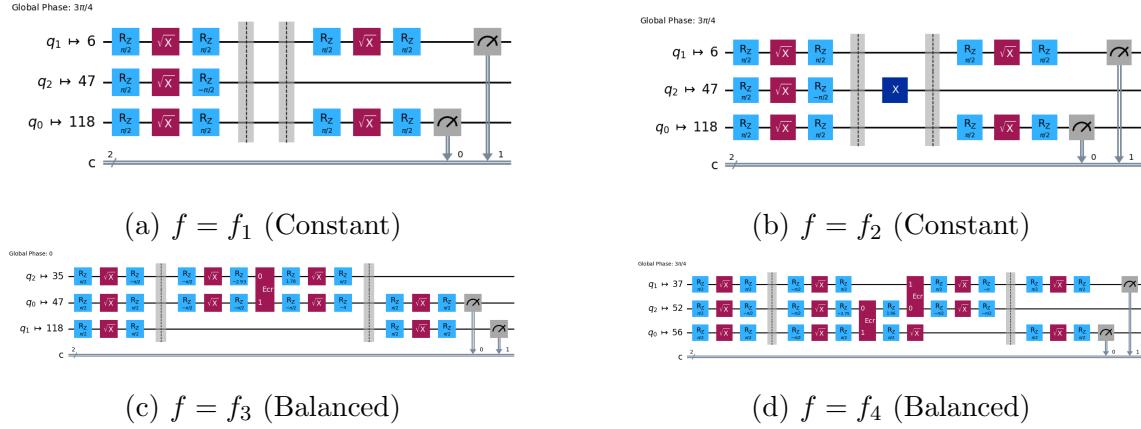


Figure 8.4: Optimized Deutsch-Jozsa Circuit implementation for possible functions along with their corresponding oracles

The Optimized circuit shown in figure 8.4 are the ones that actually get executed on a quantum computer as they are optimized for the target hardware.

### 3. Step 3: Execute using Qiskit Primitives

We are now required to run the circuit and sample the measurements on the first qubit. Thus for execution we will use the Sampler runtime primitive.

```

1 # TO run on local simulator:
2 #1. Use the StatevectorSimulator from qiskit.primitives instead
3 sampler=Sampler(backend=backend)
4 sampler.options.default_shots=10_000
5 result_const_0 = sampler.run([circuit_is_a_const_0]).result()
6 dist_const_0 = result_const_0[0].data.c.get_counts()
7

```

The output of the above code will be the distribution of the measurements on the first two qubit. Similar code can be written for all the other possible functions by just implementing the required oracle.

### 4. Step 4: Post-Process Results

Now we plot a histogram of the measurements on the first two qubits in the classical format.

```

1 from qiskit.visualization import plot_histogram
2 plot_histogram(dist_const_0)
3

```

The output of the above code will be the histogram of the distribution of the measurements on the first two qubits. Similar code can be written for all the possible functions.

The following output is produced upon running this code on ibm\_brisbane as shown in figure 8.5. The histogram plot is done for 10000 shots and results can be clearly seen as expected. The minor deviations in the results is due to the inherent errors (because of the noise) in the quantum computer and its probabilistic nature in the measurement process.

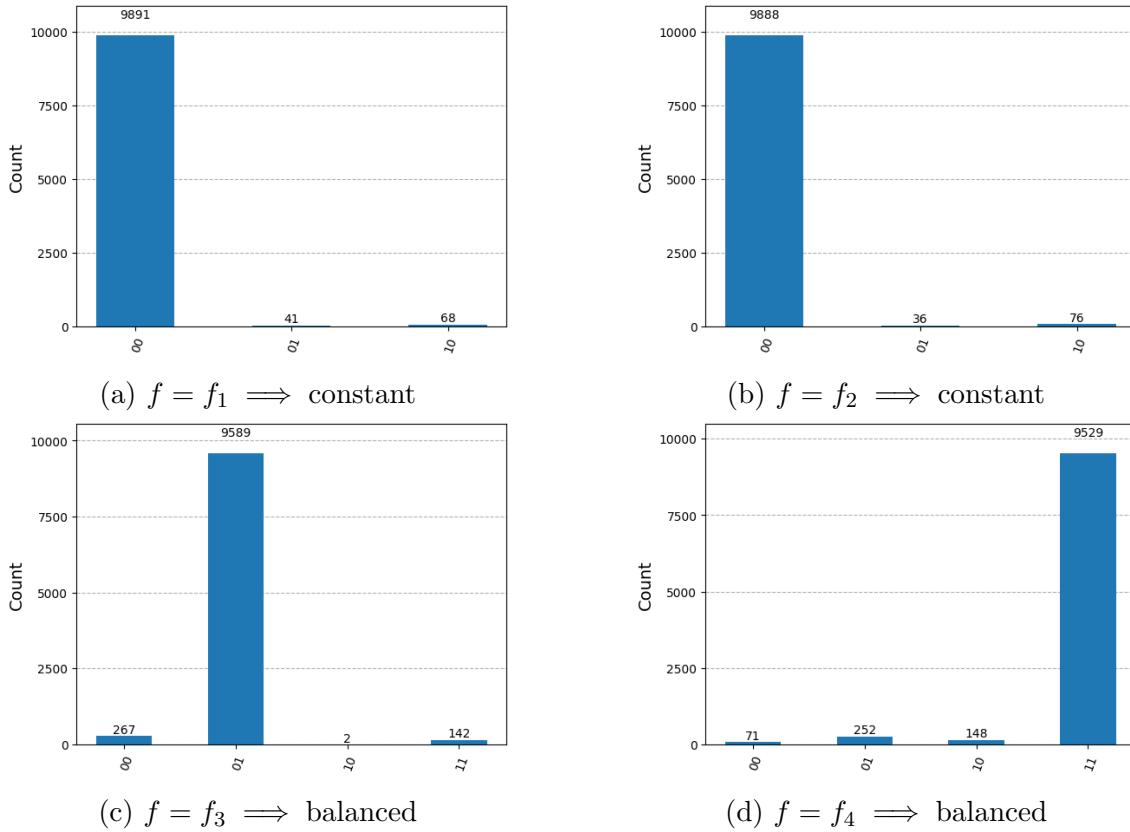


Figure 8.5: Deutsch-Jozsa Results

We can see in the figure 8.5a that the result of the measurement of the first two qubits is 00 (9891 times). Thus, we can conclusively say that the function implemented was a constant function (inside the oracle) which we know is true.

Hence, the results are expected. Similar conclusions can be made for all the other results and we can clearly see that they agree with what we implemented and expected. Here, we are able to tell whether the function was constant or balanced but not what the function was out of the different possibilities. It can also be seen that the result is  $0^n$  only in the case of constant function. It is some state orthogonal to  $0^n$  in case of balanced functions and not  $0^n$ . Note that the results might differ for different runs because of the noise in quantum computer but the answers will still follow a similar probability distribution. Similar results can also be found for four qubits circuit as shown in the figure 8.6.

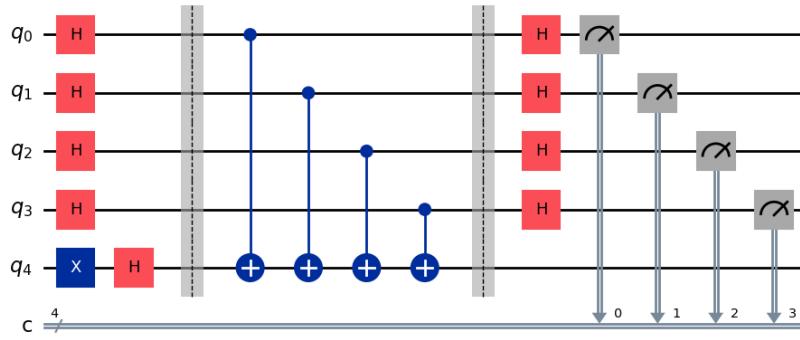


Figure 8.6: Four Qubit Example of Balanced Function

The corresponding optimal circuit is shown in figure 8.7.

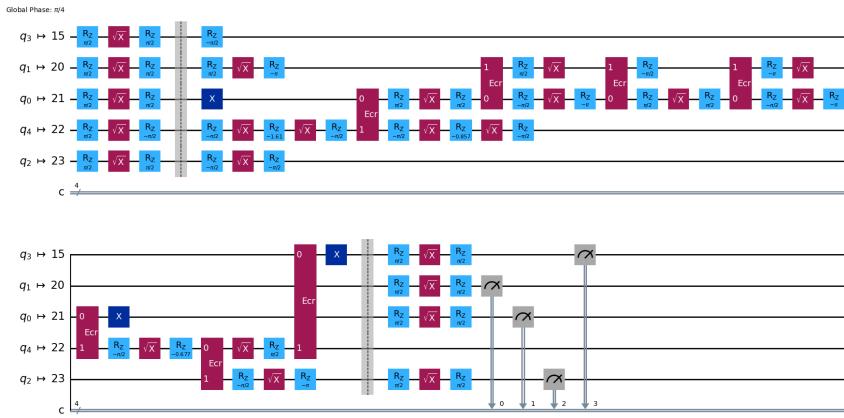


Figure 8.7: Optimized Deutsch-Jozsa Circuit for 4 qubit balanced function

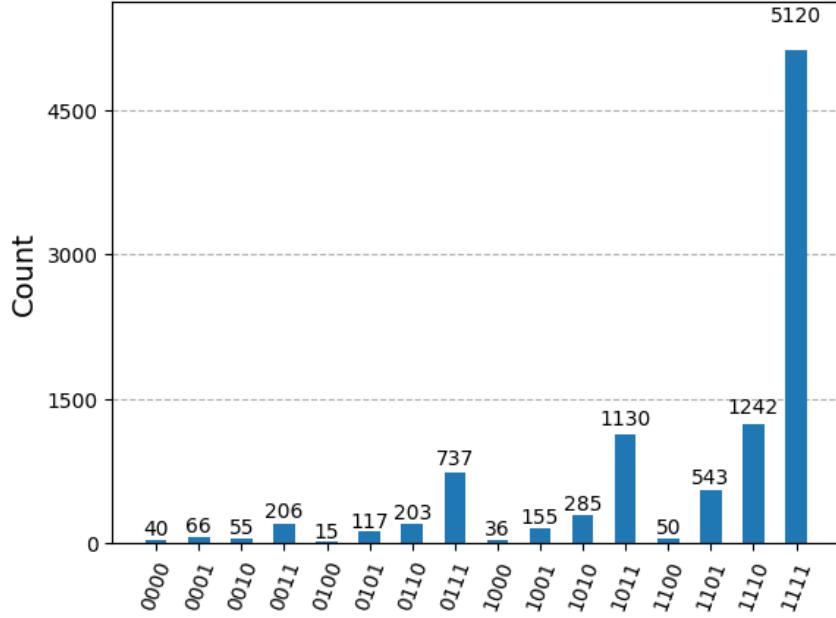


Figure 8.8: Deutsch-Jozsa results for 4 qubit balanced function

The result is shown in figure 8.8.

Similarly, circuits can be drawn for any constant or balanced function for any number of qubits and then optimized and thus, results can be obtained for any circuit.

Note that in general we do not show the oracle and instead hide the oracle. Recall from the previous results that any classical function can be implemented on quantum computer and thus, we are generally not concerned with how to implement the classical function inside the oracle, instead we focus on running the algorithm. It is always assumed that the classical oracle implemented with function will be given to us as a black-box function. Thus, the figures shown in 8.9 and the figure 8.10 are equivalent figures where in one case the function implemented is hidden inside the oracle.

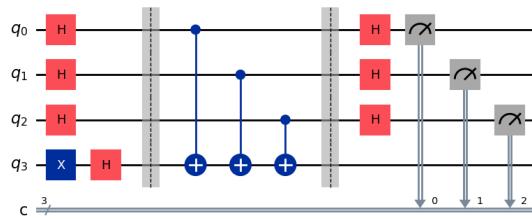


Figure 8.9: Deutsch-Jozsa Example of 3 qubit balanced function

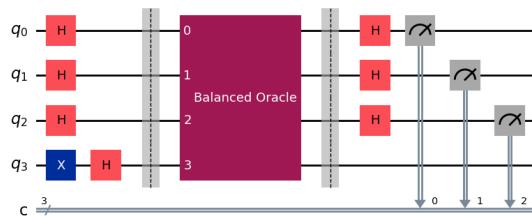


Figure 8.10: Deutsch-Jozsa Corresponding 3 qubit oracle for balanced function

# Chapter 9

## Simon's Algorithm

This problem is another one of the completely artificial problems, but now the advantage of quantum over probabilistic is quite significant: a linear number of queries will be needed by the quantum algorithm whereas an exponential number will be needed for any classical probabilistic algorithm. Also, although the problem is artificial, it is surprisingly close to a very natural and important real-world problem as we will see.

### 9.1 The Problem Definition

**Definition 9.1.1.** Given a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  which takes  $n$  input bits and returns  $n$  output bits. The function is a two - to - one function, such that  $f(x) = f(x \oplus s)$  for  $s \in \{0, 1\}^n$ . We are required to find  $s$ .

**Example 9.1.1.** Consider the function  $f : \{0, 1\}^3 \rightarrow \{0, 1\}^3$  for  $n=3$ , and let  $s = 111$ . Then, since the function is a two-to-one function i.e. produces the same output for two distinct inputs such that  $f(x) = f(x \oplus s)$ . Thus,  $f(000) = f(000 \oplus 111) = f(111)$ ,  $f(001) = f(001 \oplus 111) = f(110)$ ,  $f(010) = f(010 \oplus 111) = f(101)$ ,  $f(011) = f(011 \oplus 111) = f(100)$ ,  $f(100) = f(100 \oplus 111) = f(011)$ ,  $f(101) = f(101 \oplus 111) = f(010)$ ,  $f(110) = f(110 \oplus 111) = f(001)$ ,  $f(111) = f(111 \oplus 111) = f(000)$ . Thus, we

have:

$$\begin{aligned}f(000) &= f(111) \\f(001) &= f(110) \\f(010) &= f(101) \\f(011) &= f(100)\end{aligned}$$

Consider the following functions as an example with outputs given in tabular form:

Input	Output
000	111
001	000
010	110
011	010
100	010
101	110
110	000
111	111

Table 9.1: Example of Simon's Algorithm

It can be clearly seen that the above function is a valid function as it follows the constraints of being a two-to-one function with the above stated constraints. We can also clearly see from the table 9.1 that the function is a two-to-one function and the function is periodic with period  $s = 111$ . Thus, its a valid example function for Simon's algorithm. Now given that the function is a two-to-one function we are required to find  $s$ .

## 9.2 The Classical Solution

**On a Classical Computer:** In order to determine the period of the function we need to query the function for  $2^{n-1} + 1$  times to determine the period of the function. This is because the total number of possible inputs are  $2^n$  and since the function is a two-to-one function with a period of  $s$ , it produces the same output for half of the inputs. Thus, in order to find  $s$  classically, in the worst case scenario we are required to query the function for  $2^{n-1} + 1$  times to determine the period of the

function. Thus, the time complexity of the algorithm is  $\mathcal{O}(2^n)$  i.e. exponential time complexity. So, the best way to solve this problem is using randomized algorithms using some probabilistic approach. In that case, the running time is  $\mathcal{O}(2^{n/2})$ , which is still exponential. Note that once we know two inputs (say  $x, y$ ) with the same output. Then, we can use  $x \oplus y = s$  to find  $s$  (because  $y = x \oplus s \implies x \oplus y = x \oplus (x \oplus s) = 0 \oplus s = s$ . Because  $x \oplus x = 0$ ).

### 9.3 The Quantum Solution: Simon's Algorithm

**On a Quantum Computer:** To solve this problem we need to query the function  $\mathcal{O}(n)$  times on a Quantum Computer using the Simon's Algorithm which is a linear time complexity. The circuit for Simon's Algorithm is as shown in the figure 9.1.

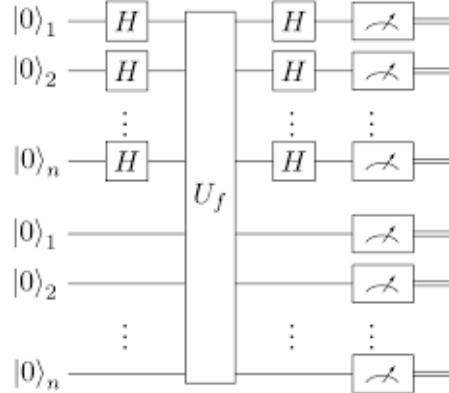


Figure 9.1: Simon's Algorithm

#### 9.3.1 Algorithm

**Analysis of Simon's Algorithm:** The input to the circuit is  $|0\rangle^{\otimes n} |0\rangle^{\otimes n}$ . Then the action of hadamard gates on the input will be:

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

Now the classical function  $f$  takes  $n$  inputs and produces  $n$  outputs and its a two-to-one function with a period  $s$ . Thus, in order to implement it on quantum computer, we use a quantum oracle  $U_f$  which implements the function  $f$  as

$U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$ . Here  $|y\rangle = |0\rangle^{\otimes n}$ , thus the action of the oracle will be  $U_f |x\rangle |0\rangle^{\otimes n} = |x\rangle |0 \oplus f(x)\rangle = |x\rangle |f(x)\rangle$ . Thus, the action of the oracle will be:

$$U_f \left( \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |0\rangle^{\otimes n} \right) = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle$$

Now we perform a measurement on the second registers. Now the second registers are in some superposition and upon measurement will collapse to some classical values. Let the output of the second register be  $f(z)$ . Because of this the first register will have only those values which are mapped to  $f(z)$ .

Thus, the state of the system will be:

$$\frac{|z\rangle + |z \oplus s\rangle}{\sqrt{2}} |f(z)\rangle$$

Now applying hadamard gate on the first n qubits will be:

$$\begin{aligned} (H^{\otimes n} \otimes I) \left( \frac{|z\rangle + |z \oplus s\rangle}{\sqrt{2}} |f(z)\rangle \right) &= \frac{1}{\sqrt{2}} [(H^{\otimes n} |z\rangle + H^{\otimes n} |z \oplus s\rangle) |f(z)\rangle] \\ &= \frac{1}{\sqrt{2}} \left[ \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{z \cdot y} |y\rangle + \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{(z \oplus s) \cdot y} |y\rangle \right] |f(z)\rangle \\ &= \frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} ((-1)^{z \cdot y} + (-1)^{(z \oplus s) \cdot y}) |y\rangle |f(z)\rangle \\ &= \frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} (-1)^{z \cdot y} (1 + (-1)^{s \cdot y}) |y\rangle |f(z)\rangle \end{aligned}$$

Now, there are two possible cases:

- **Case 1:**  $y \cdot s = 1$ . Then the amplitude of  $y$  will be zero and thus that value won't be taken in the superposition.
- **Case 2:**  $y \cdot s = 0$ . Then the amplitude of that corresponding  $y$  will be 2 and thus will be in the superposition.

Thus, the superposition will consist of only those values of  $y$  for which  $y \cdot s = 0$ . We can use this fact to run the circuit  $n$  times and attain a system of  $n$  linearly independent equations. We can then solve these equations to find  $s$ .

**Example 9.3.1.** Consider the function  $f : \{0, 1\}^4 \rightarrow \{0, 1\}^4$  as shown in the table 9.2.

Input	Output
0000,1001	1111
0001, 1000	0001
0010,1011	1110
0011,1010	1101
0100,1101	0000
0101,1100	0101
0110,1111	1010
0111,1110	1001

Table 9.2: Function

Here,  $s = 1001$ . Now, say the above shown function has been implemented in a quantum oracle. We are required to find  $s$  for the function using Simon's algorithm.

Now at the input the state of the system will be:

$$|0^4\rangle |0^4\rangle = |0000\rangle |0000\rangle$$

Now we apply hadamard gate on the first register. Thus, the state of the system will now be:

$$\begin{aligned} (H^{\otimes 4} \otimes I^{\otimes 4})(|0000\rangle \otimes |0000\rangle) &= \frac{1}{\sqrt{2^4}} \sum_{x \in \{0,1\}^4} |x\rangle |0000\rangle \\ &= \left( \frac{1}{4} \sum_{x \in \{0,1\}^4} |x\rangle \right) \otimes |0000\rangle \\ &= \left( \frac{1}{4} \sum_{x \in \{0,1\}^4} |x\rangle |0000\rangle \right) \end{aligned}$$

Now, we apply the oracle on the above state ( $U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$ ). Thus, the state of the system will be:

$$U_f \left( \frac{1}{4} \sum_{x \in \{0,1\}^4} |x\rangle |0000\rangle \right) = \frac{1}{4} \sum_{x \in \{0,1\}^4} U_f |x\rangle |0000\rangle$$

$$= \frac{1}{4} \sum_{x \in \{0,1\}^4} |x\rangle |0000 \oplus f(x)\rangle = \frac{1}{4} \sum_{x \in \{0,1\}^4} |x\rangle |f(x)\rangle$$

$$\frac{(|0000\rangle |f(0000)\rangle + |0001\rangle |f(0001)\rangle + \dots + |1110\rangle |f(1110)\rangle + |1111\rangle |f(1111)\rangle)}{4}$$

Thus, upon substituting the corresponding values from the table 9.2 we get, since  $n=4$  we will get 16 terms:

$$\begin{aligned} \frac{1}{4} [ & |0000\rangle |1111\rangle + |0001\rangle |0001\rangle + |0010\rangle |1110\rangle + |0011\rangle |1101\rangle + |0100\rangle |0000\rangle + |0101\rangle |0101\rangle \\ & |0110\rangle |1010\rangle + |0111\rangle |1001\rangle + |1000\rangle |0001\rangle + |1001\rangle |1111\rangle + |1010\rangle |1101\rangle + |1011\rangle |1110\rangle \\ & + |1100\rangle |0101\rangle + |1101\rangle |0000\rangle + |1110\rangle |1001\rangle + |1111\rangle |1010\rangle ] \end{aligned}$$

Now, we perform a measurement on the second register thus the superposition state of the second register will collapse to some classical state. Let the output of the second register be  $f(z)$  which comes out as, say  $f(z) = 1010$ . Thus, the state of the system now will be only those values which produce 1010 in the second register as output. Thus, the state of the system will be:

$$\frac{|0110\rangle + |1111\rangle}{\sqrt{2}} \otimes |1010\rangle$$

Now we apply the Hadamard gate on the first register. Thus, the state of the system will be:

$$(H^{\otimes 4} \otimes I^{\otimes 4}) \left( \frac{|0110\rangle + |1111\rangle}{\sqrt{2}} \otimes |1010\rangle \right)$$

Using the property that the Hadamard gate when applied to any input gives  $H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle$ , we get:

$$= \frac{H^{\otimes 4}|0110\rangle + H^{\otimes 4}|1111\rangle}{\sqrt{2}} \otimes |1010\rangle$$

Again applying the general formula for Hadamard gate  $H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle$ , and simplifying (recall that we are required to consider only those values where  $y \cdot s = 0$  and the amplitude in that case is 2. For the cases where the  $y \cdot s = 1$  the amplitude for that corresponding  $|y\rangle$  will be zero. Thus, we will get only those value where  $y \cdot s = 0$ ), we get:

$$\frac{|0000\rangle - |0010\rangle - |0100\rangle + |0110\rangle + |1001\rangle - |1011\rangle - |1101\rangle + |1111\rangle}{\sqrt{2^3}} \otimes |1010\rangle$$

We can also verify that  $y \cdot s = 0$ , (recall that here  $s = 1001$  and thus, only those terms in the equation survive that have  $y \cdot s = 0$ ) using the result from the above. Now when we perform the measurement on the first register we will get one of the above state with equal probability ( $= \frac{1}{8}$ ).

Now we are required to get  $n - 1 = 4 - 1 = 3$  linearly independent  $y$  to find  $s$  by forming a system of linear equations. Say we measure and upon the measurement we get the state  $|0000\rangle$ . Now state  $|0000\rangle$  is a null vector and thus we can't use it to form the system of linear equations.

Now we rerun the circuit and for the next time upon measurement we get, say  $|0010\rangle$  which works for the system of linear equations. Thus, our set of vectors for making a linear system of equation contains  $\{|0010\rangle\}$ .

Now we rerun the circuit and for the next time upon measurement we get, say  $|0100\rangle$  which works for the system of linear equations since it is independent of the previous vectors  $|0010\rangle$ . Thus, our set of vectors now contain  $\{|0010\rangle, |0100\rangle\}$ . Now we need one more linearly independent vector to find  $s$ .

Now we rerun the circuit and for the next time upon measurement we get, say  $|0110\rangle$  which does not work for the system of linear equations since it is dependent on the previous two vectors ( $|0110\rangle$  can be made using  $0100 \oplus 0010 = 0110$ ).

Now we rerun the circuit and for the next time upon measurement we get, say  $|1001\rangle$  which works for the system of linear equations since it is independent of the previous vectors  $|0010\rangle$  and  $|0100\rangle$ . Thus, our set of vectors now contains  $\{|0010\rangle, |0100\rangle, |1001\rangle\}$ . Now we have 3 linearly independent vectors and thus we can form a system of linear equations to find  $s$ .

Now, we know that each of the linearly independent vector in the set  $\{|0010\rangle, |0100\rangle, |1001\rangle\}$  is a solution to the equation  $y \cdot s = 0$ . Thus, we can form a system of linear equations as shown below:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Upon, solving this system of linear equations we get:

$$\begin{aligned} s_3 &= 0 \\ s_2 &= 0 \\ s_1 + s_4 &= 0 \end{aligned}$$

Now, we can choose one of the values of  $s_1$  or  $s_4$ . Let us choose  $s_4$ . Now,  $s_4 = 0$  or  $s_4 = 1$ . Let us choose  $s_4 = 0$  then we get,  $s_1 = 0$  which gives a trivial solution

of  $s = 0000$ . Let  $s_4 = 1$  then we get,  $s_1 = -1 \implies (-1)mod2 = 1$ . Thus, we get  $s = 1001$  which is the correct solution.

# Chapter 10

## Bernstein-Vazirani Algorithm

# Chapter 11

## Grover's Search Algorithm

It is the second most famous quantum algorithm after Shor's algorithm. It doesn't prove exponential speed up, but only a quadratic speedup, yet it is much more widely applicable than Shor's algorithm.

### 11.1 The Problem Definition

**Definition 11.1.1. The Search Problem** For  $N = 2^n$ , we are given an arbitrary  $x \in \{0, 1\}^n$ . Suppose we have a function

$$f : \{0, 1\}^n \rightarrow \{0, 1\}$$

that is implemented by a reversible transformation (Oracle)  $B_f$  in the usual way:

$$B_f |x\rangle |a\rangle = |x\rangle |a \oplus f(x)\rangle$$

for all  $x \in \{0, 1\}^n$  and  $a \in \{0, 1\}$ . (We can build a quantum circuit for  $B_f$  given a Boolean circuit for  $f$  at a cost linear in size of boolean circuit). The problem of search is to simply find a string  $x \in \{0, 1\}^n$  such that  $f(x) = 1$ , or to conclude that no such  $x$  exists if  $f$  is identically 0.

This is also called as unstructured search problem because  $f$  is arbitrary i.e. there is no promise and we can't rely on it have a structure that makes finding solutions easy.

*Remark.* Here, we assume that the we can easily evaluate this function  $f$  (function can be computed efficiently). That is, maybe we have a boolean circuit whose size is  $\text{poly}(n)$ . This does not restrict the function to be simple. It can be complex and may not have a simple form. This is a *NP-complete* problem.

## 11.2 The Classical Solution

Note that the searching problem is completely unstructured. There are no promises on the function  $f$ , so it is not possible to use binary search or any other fast searching methods to efficiently solve the problem classically. The best classical algorithm for solving the above search problem is Linear search (a **deterministic algorithm**) and in the worst case it would require  $N = 2^n$  queries to the black box (to distinguish the case where  $f$  is identically 0 from any of the cases where there is a single  $x$  for which  $f(x) = 1$ ). Thus,  $\mathcal{O}(N)$  time complexity. (Here we have assumed that the cost of evaluating  $f$  is the one that dominates the computation and hence we talk about the number of queries to the function  $f$  in order to evaluate the time complexity.)

Probabilistically, a best strategy for an algorithm that makes  $k$  queries is to simply choose  $k$  distinct values of  $x$  and to query the black-box at these  $k$  values. In the case that there is a single value of  $x$  for which  $f(x) = 1$ , and we require that our algorithm succeeds in finding this  $x$  with probability at least  $1 - \epsilon$ , then we must have  $1 - \frac{k}{2^n} \leq \epsilon$ . This implies  $k \geq (1 - \epsilon)2^n$ , so for constant error we need  $k = \Omega(2^n) = \omega(N)$  queries to solve the problem.

In contrast, the Grover's algorithm will solve the problem using  $\mathcal{O}(\sqrt{2^n})$  i.e.  $\mathcal{O}(\sqrt{N})$  queries and  $\mathcal{O}(\sqrt{N} \log N)$  other gates (the number of gates can be reduced a bit further).

### Important Note

The problem can be thought of as a table of size  $N$ , where exactly one element has value 1, and all others are 0. Searching an item in an unsorted table or array of size  $N$  costs a classical computer  $\mathcal{O}(N)$  running time. If  $N$  is large, this is like searching for a needle in a haystack. This Quantum algorithm for search was proposed by Lov Grover in 1995 that consults the table only  $\mathcal{O}(\sqrt{N})$  times.

In contrast to algorithms like quantum factoring which provide exponential speedups, the search algorithm only provides a quadratic improvement. The same technique used in this algorithm can be used to speedup algorithms for NP-complete problems.

It is natural to wonder whether there are even faster quantum algorithms for search. However, it turns out that the quadratic speedup is optimal. This was proved in 1994. Any quantum algorithm for search must consult the table at least some constant times  $\sqrt{N}$  times. In this, we have assumed *unique search problem* i.e. This is all assuming that there is only one element that has value 1 and all others have value 0. In case of *multiple solutions* i.e. if there are more than one elements with value 1, then the Grover's algorithm can be modified, and in that case the Complexity becomes  $\mathcal{O}(\sqrt{\frac{N}{M}})$ , where  $M$  is the number of elements with value 1 and  $N$  is the total number of elements ( $N = 2^n$ ).

## 11.3 The Quantum Solution: Grover's Algorithm

Here, we assume the problem to be a Query problem. Thus, the boolean function  $f$  implemented on a classical circuit can be converted to a Quantum circuit for implementing a Query operation using the concepts of reversible computation as explained in section 5.12.

### 11.3.1 Idea of the Algorithm

The best way to describe the Grover's Algorithm is to describe it geometrically. Say we define two sets  $A$  and  $B$  as follows:

$$\begin{aligned} A &= \{x \in \{0,1\}^n : f(x) = 1\} \\ B &= \{x \in \{0,1\}^n : f(x) = 0\} \end{aligned}$$

Think of  $A$  as the set of "good" strings  $x \in \{0, 1\}^n$ ; the goal of the algorithm is to find one of these strings. The set  $B$  contains all of the "bad" strings  $x \in \{0, 1\}^n$  that do not satisfy the search criterion. Let

$$a = |A| \quad b = |B|; \quad a + b = N$$

where  $|\cdot|$  denotes cardinality of the sets (i.e. number of elements in the set), with  $a \neq 0$  and  $b \neq 0$  (The case with  $a = 0$  or  $b = 0$  will be considered separately). Although the analysis is for the general case, it might be easier to imagine an interesting case of the problem where  $a$  is very small (say  $a = 1$ ) and therefore  $b$  is very large (close to  $N = 2^n$ , say  $b = N - 1$ ). Note that for each  $x \in \{0, 1\}^n$  it will be either in  $A$  or in  $B$  but not both since the output corresponding to the input to the function is either 0 or 1. Thus, an arbitrary  $x$  will be either in  $A$  or in  $B$ . In other words, the sets  $A$  and  $B$  are mutually exclusive sets. Next we define,

$$\begin{aligned} |A\rangle &= \frac{1}{\sqrt{a}} \sum_{x \in A} |x\rangle \\ |B\rangle &= \frac{1}{\sqrt{b}} \sum_{x \in B} |x\rangle \end{aligned}$$

which means  $|A\rangle$  is the superposition of all the input states to the function  $f$  which yield output 1 and similarly,  $|B\rangle$  is the superposition of all the input states to the function  $f$  which output 0. Now since sets  $A$  and  $B$  were mutually exclusive and each  $x$  lies either in  $A$  or  $B$ . Clearly  $|A\rangle$  and  $|B\rangle$  are orthogonal i.e.  $\langle A|B \rangle = 0$  because each of the basis state  $|x\rangle$  is either in  $A$  or in  $B$ , thus its corresponding component in the other set is 0, thus their inner product will result in 0. Hence,  $|A\rangle$  and  $|B\rangle$  are orthogonal.

The two vectors  $|A\rangle$  and  $|B\rangle$  form a two-dimensional subspace geometrically. Now consider an equal superposition state of all the possible  $|x\rangle$  i.e.  $|h\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle$ . Note that the two-dimensional subspace formed by the two orthogonal vectors  $|A\rangle$  and  $|B\rangle$  contains  $|h\rangle$  i.e.  $|h\rangle$  can be expressed as a linear combination of the basis

vectors of the two-dimensional subspace as follows:

$$\begin{aligned}
 |\hbar\rangle &= \frac{1}{\sqrt{N}} \sum_x |x\rangle \\
 &= \frac{1}{\sqrt{N}} \left( \sum_{x \in A} |x\rangle + \sum_{x \in B} |x\rangle \right) \\
 &= \frac{\sqrt{a}}{\sqrt{N}} \frac{1}{\sqrt{a}} \sum_{x \in A} |x\rangle + \frac{\sqrt{b}}{\sqrt{N}} \frac{1}{\sqrt{b}} \sum_{x \in B} |x\rangle \\
 |\hbar\rangle &= \frac{\sqrt{a}}{\sqrt{N}} |A\rangle + \frac{\sqrt{b}}{\sqrt{N}} |B\rangle
 \end{aligned}$$

Considering for the purpose of imagination one could think of  $a \ll b$  so that the search problem is like finding a needle in a haystack. In other words, the component of  $|\hbar\rangle$  on  $|A\rangle$  is much smaller than the component of  $|\hbar\rangle$  on  $|B\rangle$ . Thus, geometrically in space one could imagine this as  $|\hbar\rangle$  being very close to  $|B\rangle$  then  $|A\rangle$  as depicted in the following figure 11.1

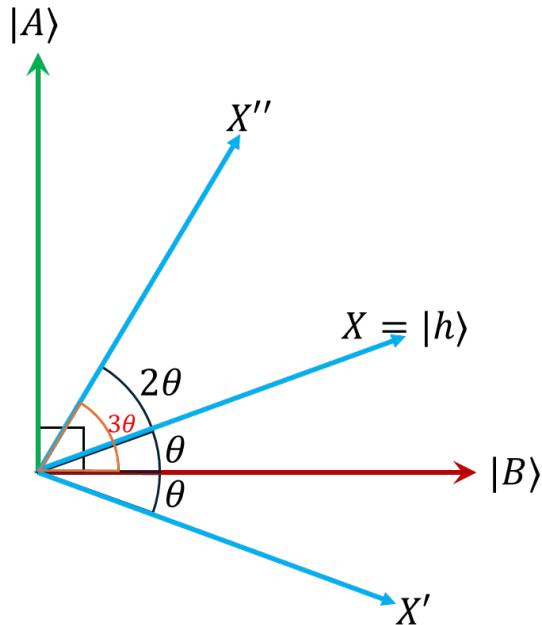


Figure 11.1: Geometric Visualization

In the figure 11.1 it can be clearly seen that the  $|A\rangle$  and  $|B\rangle$  are orthogonal. Say

$X = |h\rangle$  makes an angle  $\theta$  with  $|B\rangle$  (Note that for the purpose of easier imagination as said in the case where  $a \ll b$ , (say  $a = 1, b = N - 1$ ) which is mostly the case, i.e. when the number of good strings to be searched for is much less than the bad strings, the component of  $|h\rangle$  on  $|A\rangle$  ( $= \frac{\sqrt{a}}{\sqrt{N}}$ ) will be much less than the component of  $|B\rangle$  on  $|h\rangle$  ( $= \frac{\sqrt{b}}{\sqrt{N}}$ ) since  $a \ll b$ . Thus, as shown in the figure 11.1,  $|h\rangle$  is much closer to  $|B\rangle$  than  $|A\rangle$ .)

Note that it is easy to create the state  $|h\rangle$  since it is simply an equal superposition of all the states. Thus can be created by applying  $H^{\otimes n}$  to  $|0^n\rangle$ . From figure 11.1 we are required to make a series of rotations/unitary transformations such that eventually  $|h\rangle$  gets closer to  $|A\rangle$  (or becomes  $|A\rangle$ ), thus achieving the purpose of finding the superposition of strings  $x \in \{0, 1\}^n$  whose input to the function yields output 1. Then, upon measurement of this state we get one of the required strings whose input to the function yields output 1.

The Unitary transformations (recall that they are simply rotations or reflections in the complex Plane i.e. Hilbert Space) required to perform this are a series of reflections as follows (Starting with  $|X\rangle = |h\rangle$ ):

1. Reflect  $X$  about  $|B\rangle$  which as shown in the figure 11.1 becomes  $|X'\rangle$ . Say this is denoted by the Unitary matrix  $R_{|B\rangle}$ . Thus,  $|X'\rangle = R_{|B\rangle} |X\rangle$
2. Now reflect  $|X'\rangle$  about  $X$  which as shown in the figure 11.1 becomes  $|X''\rangle$ . Say this is denoted by the Unitary matrix  $R_{|X\rangle}$ . Thus,  $|X''\rangle = R_{|X\rangle} |X'\rangle$

After each iteration we put  $|X\rangle = |X''\rangle$  and repeat the above two steps. Thus, the angle of  $|X''\rangle$  with  $|B\rangle$  is  $3\theta$  after doing this reflections for the first time. Hence, the  $|X''\rangle$  obtained in the second step is now clearly closer to  $|A\rangle$ . Thus, performing the above steps repeatedly will bring the  $|X\rangle$  closer and closer to  $|A\rangle$  with each iteration as it moves farther and farther from  $|B\rangle$ . Once the state  $|X''\rangle$  is close enough to  $|A\rangle$ , measuring the state will result in the required search string with a very high probability. Thus, till now what we have done is the following:

$$|0^n\rangle \xrightarrow{H^{\otimes n}} |h\rangle$$

Then we repeat the following unitary transformation a finite number of times ( $\mathcal{O}(\sqrt{N})$  times) starting with  $|X\rangle = |h\rangle$

$$|X\rangle \xrightarrow{R_{|B\rangle}} |X'\rangle \xrightarrow{R_{|X\rangle}} |X''\rangle$$

Then, finally we make a measurement on  $|X''\rangle$  (since it gets very close to  $A$  as shown above) and the result of that measurement is one the strings whose input to the

function yields output 1. Thus, the required algorithm. This is the idea behind the Grover's algorithm. It would suffice if we could find the two unitary transformation matrices corresponding to the reflections and show that it does perform the task of reflection about  $|B\rangle$  and reflection about  $|X\rangle$  respectively and along with that we are required to find the optimal number of times to repeat the process. All of this is answered in the next section.

### 11.3.2 Algorithm

#### Reflection about $|B\rangle$

Recall that  $|h\rangle$  can be written as linear superposition of the basis  $|A\rangle$  and  $|B\rangle$  as follows (where both  $|A\rangle$  and  $|B\rangle$  are orthogonal):

$$|h\rangle = \frac{\sqrt{a}}{\sqrt{N}} |A\rangle + \frac{\sqrt{b}}{\sqrt{N}} |B\rangle$$

In order to reflect  $|h\rangle$  about  $|B\rangle$ , see figure 11.1, we are required to reverse the component of projection of  $|h\rangle$  on  $|A\rangle$ . Thus, the reflected vector should have the following linear superposition:

$$|X'\rangle = -\frac{\sqrt{a}}{\sqrt{N}} |A\rangle + \frac{\sqrt{b}}{\sqrt{N}} |B\rangle$$

In other words, all we need to do is flip the phase of the component in the direction of  $|A\rangle$ . For a general  $|X\rangle$  which arises in some intermediate step of the iteration, (which will be a linear superposition of  $|A\rangle$  and  $|B\rangle$ ) we are required to reverse the phase on the components in the direction of  $|A\rangle$  keeping the components in the direction of  $|B\rangle$  unchanged. In order to achieve this, we use **Phase Query Gate**. Consider the following unitary transformation on n qubits:

$$Z_f |x\rangle = (-1)^{f(x)} |x\rangle$$

This is also called a *phase kickback*. This transformation would take in the superposition state and reverse the phase of only those components which are in  $|A\rangle$  (i.e. whose  $f(x) = 1$ ) using linearity property of distributive action of a matrix on sum of vectors (superposition state). We now study the corresponding circuit as shown in the figure 11.2 which performs this transformation.

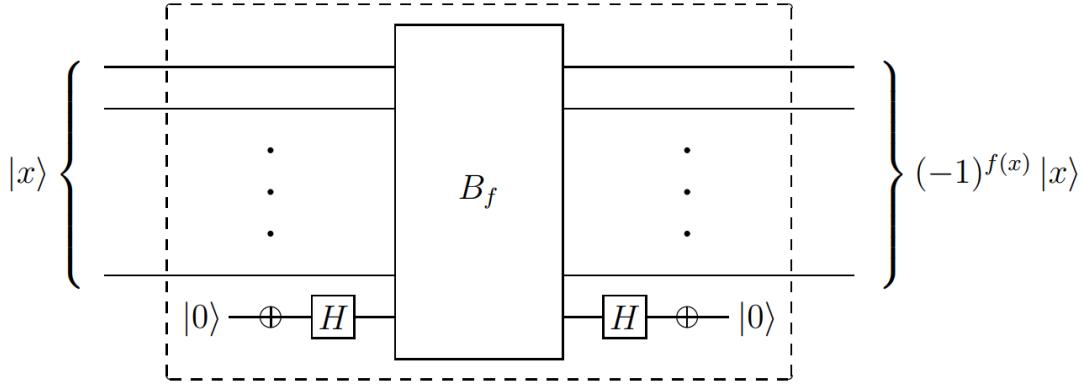


Figure 11.2: Reflection about B circuit

Here, recall that the transformation  $B_f$  is the quantum implementation of the classical function  $f$  i.e.  $B_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$ . We now analyse the circuit. The input to the circuit is  $|x\rangle |0\rangle$ . Here  $|0\rangle$  is the ancilla bit. We then apply NOT-gate on the second register:

$$|x\rangle |0\rangle \xrightarrow{I^{\otimes n} \otimes X} |x\rangle |1\rangle$$

Then we apply a Hadamard gate on the output on the second register, thus we get,

$$|x\rangle |1\rangle \xrightarrow{I^{\otimes n} \otimes H} |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Now we use the Oracle  $B_f$  on this,

$$B_f |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} B_f |x\rangle |0\rangle - \frac{1}{\sqrt{2}} B_f |x\rangle |1\rangle = \frac{1}{\sqrt{2}} |x\rangle |0 \oplus f(x)\rangle - \frac{1}{\sqrt{2}} |x\rangle |1 \oplus f(x)\rangle$$

This can be further simplified as follows:

$$\frac{1}{\sqrt{2}} |x\rangle |f(x)\rangle - \frac{1}{\sqrt{2}} |x\rangle |\overline{f(x)}\rangle = |x\rangle \frac{|f(x)\rangle - |\overline{f(x)}\rangle}{\sqrt{2}} = (-1)^{f(x)} |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Then we perform the uncomputation on the second register by applying a Hadamard gate and then a NOT gate as shown in the figure 11.2.

$$(-1)^{f(x)} |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \xrightarrow{I^{\otimes n} \otimes H} (-1)^{f(x)} |x\rangle |1\rangle \xrightarrow{I^{\otimes n} \otimes X} (-1)^{f(x)} |x\rangle |0\rangle$$

Now, ignoring the ancilla bit we have thus achieved the following transformation

$$Z_f |x\rangle = (-1)^{f(x)} |x\rangle$$

which was the required transformation for the reflection of  $|X\rangle$  over  $|B\rangle$  done by reversing the phase of the components in the direction of  $|A\rangle$ . Note that we were required to perform one evaluation of  $B_f$  implemented in  $X_f$  to reflect about  $|B\rangle$ . Thus, what we have achieved through this Oracle as a Unitary operator can be seen from the fact that consider a  $|X\rangle$  as some linear superposition of  $|A\rangle$  and  $|B\rangle$  (recall,  $|A\rangle$  and  $|B\rangle$  are orthogonal and  $|X\rangle$  lies in the two-dimensional subspace spanned by  $|A\rangle$  and  $|B\rangle$ ) , say  $|X\rangle = r|A\rangle + s|B\rangle$  for some  $r, s$ . Then upon applying the operator/Oracle  $Z_f$  on this  $|X\rangle$  we get,

$$\begin{aligned} Z_f |X\rangle &= rZ_f |A\rangle + rZ_f |B\rangle \\ &= rZ_f \frac{1}{\sqrt{a}} \sum_{x \in A} |x\rangle + sZ_f \frac{1}{\sqrt{b}} \sum_{x \in B} |x\rangle \\ &= r \frac{1}{\sqrt{a}} \sum_{x \in A} Z_f |x\rangle + s \frac{1}{\sqrt{b}} \sum_{x \in B} Z_f |x\rangle \\ &= r \frac{1}{\sqrt{a}} \sum_{x \in A} -|x\rangle + s \frac{1}{\sqrt{b}} \sum_{x \in B} |x\rangle \\ &= -r|A\rangle + s|B\rangle \end{aligned}$$

As can be seen that the component of  $|X\rangle$  along  $|A\rangle$  is reversed. Thus, we achieve the reflection about  $|B\rangle$ .

### Reflection about $|X\rangle$

For the reflection about  $|X\rangle$ , we use the Diffusion operator  $D$  (assume  $N = 2^n$ ), which works as follows. First, apply  $H^{\otimes n}$ , which maps  $|X\rangle \rightarrow |00\dots 0\rangle$ . then reflect around  $|00\dots 0\rangle$  this is accomplished by the circuit  $U_g$ , where  $g$  is a function such that  $g(00\dots 0) = 0$  and  $g(x) = 1$  for  $x \neq 00\dots 0$ . Finally, apply  $H^{\otimes n}$  to return to the original basis. (Note that this is simply a reflection around the zero vector in the Hadamard basis). This is the general idea of Reflection about  $|X\rangle$ . In order to understand this operation better we get into the details as follows. Consider an operator  $Z_0$  as follows:

$$Z_0 = I - 2|0^n\rangle \langle 0^n|$$

Thus,

$$D = H^{\otimes n} Z_0 H^{\otimes n} = H^{\otimes n} (I - 2|0^n\rangle \langle 0^n|) H^{\otimes n} = I - 2|h\rangle \langle h|$$

This is also the same as having a phase query gate  $Z_{OR}$  ( $= -Z_0$ ) for the n-bit OR function:

$$OR(x) = \begin{cases} 0, & x = 0^n \\ 1, & x \neq 0^n \end{cases}$$

for all  $x \in \{0, 1\}^n$ .

$$Z_{OR}(x) = \begin{cases} |x\rangle & x = 0^n \\ -|x\rangle & x \neq 0^n \end{cases}$$

for all  $x \in \{0, 1\}^n$ . Note that  $Z_{OR} = -Z_0 = 2|0^n\rangle\langle 0^n| - I$ . Note that this does not depend on function  $f$  and thus requires no queries. To implement this on a quantum circuit we convert the classical implementation of a n-bit OR boolean circuit. We now make the following propositions:

**Proposition 11.3.1.** *The Diffusion operator D has two properties:*

1. *It is Unitary and can be efficiently realised.*
2. *It can be seen as an "inversion about the mean".*

*Proof.* 1. To prove that D is unitary it suffices to prove that  $H^{\otimes n}$  and  $Z_0$  is Unitary. Clearly,  $H^{\otimes n}$  is unitary since it is a Hadamard Gate tensored product n times. Now for proving that  $Z_0$  is unitary. We do as follows:

$$\begin{aligned} Z_0^\dagger &= (I - 2|0^n\rangle\langle 0^n|)^\dagger \\ &= I^\dagger - 2(|0^n\rangle\langle 0^n|)^\dagger \\ &= I - 2|0^n\rangle\langle 0^n| = Z_0 \end{aligned}$$

Thus,  $Z_0$  is Hermitian. Now doing  $Z_0^\dagger Z_0$ , we get,

$$\begin{aligned} Z^\dagger Z_0 &= Z_0 Z_0^\dagger = (I - 2|0^n\rangle\langle 0^n|)(I - 2|0^n\rangle\langle 0^n|) \\ &= I - 2|0^n\rangle\langle 0^n| - 2|0^n\rangle\langle 0^n| + 4|0^n\rangle\langle 0^n| \\ &= I \end{aligned}$$

Thus,  $Z_0$  is Unitary. Hence  $D = H^{\otimes n} Z_0 H^{\otimes n}$  is a product of Unitary matrices and hence itself is a Unitary Matrix. Observe that D is expressed as the product of three Unitary matrices (two hadamard Matrices separated by a conditional phase shift matrix). Therefore, D is also Unitary. regarding the implementation, both the Hadamard and the conditional phase shift transforms can be efficiently realized within  $\mathcal{O}(n)$  gates.

2. Consider  $D$  operating on a vector  $|\alpha\rangle$  to generate another vector  $|\beta\rangle$ .

$$D \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_i \\ \vdots \\ \alpha_N \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_i \\ \vdots \\ \beta_N \end{bmatrix}$$

Let  $\mu = \frac{1}{N} \sum_j \alpha_j$  be the mean amplitude, then the expression  $2\mu - \alpha_i$  describes a reflection of  $\alpha_i$  about mean. This might be easier to see by writing it as  $\mu + (\mu - \alpha_i)$ . In order to realise that the diffusion operator does perform this task, let us write  $D$  in matrix form ( $N = 2^n$ ):

$$D = I - 2|h\rangle\langle h|$$

$$= \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix} - \frac{2}{N} \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \\ 1 & 1 & \dots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \dots & 1 & 1 \\ 1 & 1 & \dots & 1 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 - \frac{2}{N} & -\frac{2}{N} & \dots & -\frac{2}{N} & -\frac{2}{N} \\ -\frac{2}{N} & 1 - \frac{2}{N} & \dots & -\frac{2}{N} & -\frac{2}{N} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -\frac{2}{N} & -\frac{2}{N} & \dots & 1 - \frac{2}{N} & -\frac{2}{N} \\ -\frac{2}{N} & -\frac{2}{N} & \dots & -\frac{2}{N} & 1 - \frac{2}{N} \end{bmatrix}$$

Thus, the amplitude of  $\beta_i = -\frac{2}{N} \sum_j \alpha_j + \alpha_i = -2\mu + \alpha_i$  can be considered an "inversion about the mean" with respect to  $\alpha_i$ .

□

Now, in order to understand how we achieve the reflection about  $|h\rangle$ , let us understand that the operator  $D = I - 2|h\rangle\langle h|$  is a reflection operator about a hyperplane orthogonal to  $|h\rangle$ . Consider the vector  $|X'\rangle$ . Now we can write  $|X'\rangle$  as a linear superposition of components along  $|h\rangle$  and components orthogonal to  $|h\rangle$  which is denoted by  $|h^\perp\rangle$  (we can do this because recall that  $|X'\rangle$  and  $|h\rangle$  both lie in the two-dimensional subspace spanned by the orthogonal basis  $|A\rangle$  and  $|B\rangle$ ). Thus, when we say component orthogonal to  $|h\rangle$ , we mean that  $|h^\perp\rangle$  which lies in this

two-dimensional subspace, since, the component of  $|X'\rangle$  which is orthogonal to  $|h\rangle$  will obviously lie in the plane spanned by  $|A\rangle$  and  $|B\rangle$ ). Thus,  $|X'\rangle = p|h\rangle + q|h^\perp\rangle$  for some  $p, q$ . Now we apply the operator  $D$  on  $|X'\rangle$ , we get,

$$\begin{aligned} D|X'\rangle &= pD|h\rangle + qD|h^\perp\rangle \\ &= p(I - 2|h\rangle\langle h|)|h\rangle + q(I - 2|h\rangle\langle h|)|h^\perp\rangle \\ &= p(|h\rangle - 2|h\rangle\langle h|h\rangle) + q(|h^\perp\rangle - 2|h\rangle\langle h|h^\perp\rangle) \\ &= p|h\rangle - 2p|h\rangle + q|h^\perp\rangle \\ &= -p|h\rangle + q|h^\perp\rangle \end{aligned}$$

Thus, the component along  $|h\rangle$  is reversed while the component along the hyperplane orthogonal to  $|h\rangle$  (i.e. along  $|h^\perp\rangle$ ) is preserved. Hence, it's a reflection of  $|X'\rangle$  about a plane orthogonal to  $|h\rangle$ . Now, we simply reverse the sign i.e. use  $-D$  as operator. Thus, we get,

$$-D|X'\rangle = p|h\rangle - q|h^\perp\rangle$$

which preserves the component along  $|h\rangle$  and reverses the component along the plane orthogonal to  $|h\rangle$  (i.e. along  $|h^\perp\rangle$ ). Hence, this operation is a reflection of  $|X'\rangle$  about  $|h\rangle$ . Thus, the operator  $-D$  is the required unitary transformation operator which performs the required reflection of  $|X'\rangle$  operation about  $|h\rangle$ .

### Grover's Rotation Operator

We have seen that using unitary operator/matrix  $Z_f$  (as an oracle) we can perform the required reflection transformation about  $|B\rangle$  and using  $-D$  operator/matrix we can perform the required reflection transformation about  $|h\rangle$ . Now, let us combine the two operations into one and define the Grover's Rotation operator as follows:

$$G = -DZ_f = -H^{\otimes n}Z_0H^{\otimes n}Z_f$$

We are now interested in the action of  $G$  on elements in the two-dimensional subspace spanned by  $\{|A\rangle, |B\rangle\}$ . The action of any matrix on any element in the subspace can be determined by examining its effect on the basis vectors of the subspace. Thus, in order to determine the action of Grover's operator  $G$ , it is sufficient to examine its effects on the basis vectors  $|A\rangle$  and  $|B\rangle$  of the two-dimensional subspace. Thus, the

action of  $G$  on  $|A\rangle$  is given as follows:

$$\begin{aligned}
 G|A\rangle &= -H^{\otimes n}Z_0H^{\otimes n}Z_f|A\rangle \\
 &= (I - 2|h\rangle\langle h|)(-Z_f)|A\rangle \\
 &= (I - 2|h\rangle\langle h|)(-Z_f)\frac{1}{\sqrt{a}}\sum_{x \in A}|x\rangle \\
 &= (I - 2|h\rangle\langle h|)\frac{1}{\sqrt{a}}\sum_{x \in A}-Z_f|x\rangle \\
 &= (I - 2|h\rangle\langle h|)\frac{1}{\sqrt{a}}\sum_{x \in A}|x\rangle \\
 &= (I - 2|h\rangle\langle h|)|A\rangle \\
 &= |A\rangle - 2|h\rangle\langle h|A\rangle
 \end{aligned}$$

Upon Substituting the value of  $|h\rangle = \sqrt{\frac{a}{N}}|A\rangle + \sqrt{\frac{b}{N}}|B\rangle$ , and simplifying we get,

$$\begin{aligned}
 &= |A\rangle - 2\left(\sqrt{\frac{a}{N}}|A\rangle + \sqrt{\frac{b}{N}}|B\rangle\right)\left(\sqrt{\frac{a}{N}}\langle A|A\rangle + \sqrt{\frac{b}{N}}\langle B|A\rangle\right) \\
 &= |A\rangle - 2\sqrt{\frac{a}{N}}\left(\sqrt{\frac{a}{N}}|A\rangle + \sqrt{\frac{b}{N}}|B\rangle\right) \\
 &= \left(1 - \frac{2a}{N}\right)|A\rangle - \frac{2\sqrt{ab}}{N}|B\rangle
 \end{aligned}$$

Note that action of  $G|A\rangle$  results in a vector in the same two-dimensional subspace spanned by  $|A\rangle$  and  $|B\rangle$ . and similarly action of operator  $G$  on  $|B\rangle$  is as follows:

$$\begin{aligned}
 G|B\rangle &= -H^{\otimes n}Z_0H^{\otimes n}Z_f|B\rangle \\
 &= -(I - 2|h\rangle\langle h|)|B\rangle \\
 &= -(|B\rangle - 2|h\rangle\langle h|B\rangle) \\
 &= -|B\rangle + 2\sqrt{\frac{b}{N}}\left(\sqrt{\frac{a}{N}}|A\rangle + \sqrt{\frac{b}{N}}|B\rangle\right) \\
 &= \frac{2\sqrt{ab}}{N}|A\rangle - \left(1 - \frac{2b}{N}\right)|B\rangle
 \end{aligned}$$

which is also a vector in the two-dimensional subspace spanned by  $|A\rangle$  and  $|B\rangle$ . Thus, action of  $G$  on some  $|X\rangle$  in the two dimensional space spanned by  $|A\rangle$  and  $|B\rangle$  (say  $|X\rangle = \alpha|A\rangle + \beta|B\rangle$ ) is  $G|X\rangle = \alpha G|A\rangle + \beta G|B\rangle$ . From the above results, we can see that  $|X\rangle$  after applying  $G$  remains in the subspace spanned by  $|A\rangle$  and  $|B\rangle$  with real coefficients. Let us represent the action of  $G$  on the subspace spanned by  $\{|A\rangle, |B\rangle\}$  as a matrix in  $|B\rangle, |A\rangle$  basis (both input and output basis) will be as follows:

$$M = \begin{bmatrix} -\left(1 - \frac{2b}{N}\right) & -\frac{2\sqrt{ab}}{N} \\ \frac{2\sqrt{ab}}{N} & \left(1 - \frac{2a}{N}\right) \end{bmatrix} = \begin{bmatrix} \frac{b-a}{N} & -\frac{2\sqrt{ab}}{N} \\ \frac{2\sqrt{ab}}{N} & \frac{b-a}{N} \end{bmatrix}$$

using  $a + b = N$ . Also, here  $|B\rangle$  is the first column and  $|A\rangle$  is the second column. Notice that

$$\begin{bmatrix} \sqrt{\frac{b}{N}} & -\sqrt{\frac{a}{N}} \\ \sqrt{\frac{A}{N}} & \sqrt{\frac{b}{N}} \end{bmatrix}^2 = \begin{bmatrix} \frac{b-a}{N} & -\frac{2\sqrt{ab}}{N} \\ \frac{2\sqrt{ab}}{N} & \frac{b-a}{N} \end{bmatrix} = M$$

Thus,  $M$  is a rotation matrix. Now, for  $\theta \in (0, \pi/2)$  is the angle that satisfies

$$\sin \theta = \sqrt{\frac{a}{N}} \quad \text{and} \quad \cos \theta = \sqrt{\frac{b}{N}}$$

then

$$R_{2\theta} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}^2 = \begin{bmatrix} \sqrt{\frac{b}{N}} & -\sqrt{\frac{a}{N}} \\ \sqrt{\frac{A}{N}} & \sqrt{\frac{b}{N}} \end{bmatrix}^2 = M$$

In other words, operator  $G$  causes a rotation by an angle  $2\theta$  in the space spanned by  $\{|A\rangle, |B\rangle\}$  for

$$\theta = \sin^{-1} \sqrt{\frac{a}{N}}$$

Now, we can write the state  $|h\rangle$  as follows show in figure 11.3:

$$|h\rangle = \sqrt{\frac{b}{N}}|B\rangle + \sqrt{\frac{a}{N}}|A\rangle = \cos \theta |B\rangle + \sin \theta |A\rangle$$

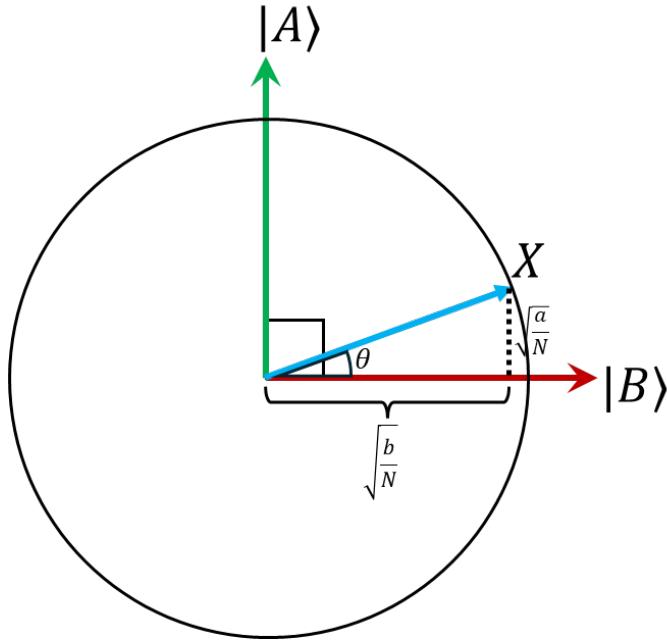


Figure 11.3: Action of G

Thus, the action of operator  $G$  on  $|X\rangle$  for  $k$  times (i.e. after  $k$  iterations), the state  $|X\rangle$  will be,

$$|X\rangle = \cos((2k + 1)\theta) |B\rangle + \sin((2k + 1)\theta) |A\rangle$$

We take  $k = \lfloor \frac{\pi}{4}\sqrt{N} \rfloor$  for the case assuming  $a = 1$ . Note that Measurement in the

**Algorithm 1** Grover's Algorithm

- 1: **Input:** Oracle  $B_f$ , number of qubits  $n$
- 2: **Output:**  $x$  such that  $f(x) = 1$
- 3: **Initialize:**  $|0^n\rangle$
- 4: Apply Hadamard gate to each qubit in  $|0^n\rangle$ ,  $|X\rangle = H^{\otimes n}|0^n\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle = |h\rangle$
- 5: **for**  $i = 1$  to  $\lfloor \frac{\pi}{4}\sqrt{N} \rfloor$  **do**
- 6:     Apply Grover's operator  $G$  on  $|X\rangle \implies G|X\rangle$
- 7:     Set  $|X\rangle = G|X\rangle$
- 8: **end for**
- 9: Measure the final state  $|X\rangle$  in Standard Basis
- 10: **return**  $x$  of the measured state

standard basis may reveal one of the candidate solutions based on the number of times we iterate the for loop. The Grover's circuit corresponding to above implementation is as shown in the figure 11.4.

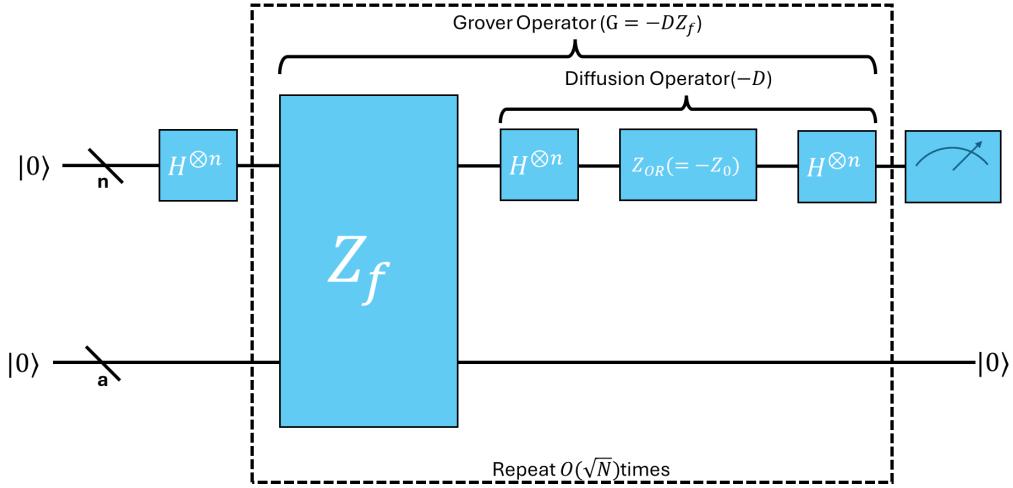


Figure 11.4: Grover's Circuit

Note that both the unitary operations  $Z_f$  which is a reflection about  $|B\rangle$  and reflection about  $|h\rangle$  are reflection operations/transformation and hence, the determinant of those two unitary operations is  $-1$ . When we compose two reflections, we obtain a rotation by twice the angle between the lines of reflection.

### 11.3.3 Example

For example, consider the function  $f : \{0, 1\}^3 \rightarrow \{0, 1\}$  as shown in the table 11.1.

Input	Output
000	0
001	0
010	0
011	1
100	1
101	0
110	0
111	0

Table 11.1: Example of Grover's Algorithm

Thus, the set  $A = \{011, 100\}$  and  $B = \{000, 001, 010, 101, 110, 111\}$ . Let the Cardinality i.e. the number of elements in the set represented as  $|A| = a = 2$  and  $|B| = b = 6$ . Here, in this example,  $a = 2$  and  $b = 6$  and  $n = 3$ . Now, we define the vector  $|A\rangle$  as:

$$|A\rangle = \frac{1}{\sqrt{a}} \sum_{x \in A} |x\rangle = \frac{1}{\sqrt{2}}(|011\rangle + |100\rangle)$$

Similarly, we define the vector  $|B\rangle$  as:

$$|B\rangle = \frac{1}{\sqrt{b}} \sum_{x \in B} |x\rangle = \frac{1}{\sqrt{6}}(|000\rangle + |001\rangle + |010\rangle + |101\rangle + |110\rangle + |111\rangle)$$

Thus, it can be clearly seen that  $|A\rangle$  and  $|B\rangle$  are orthogonal as their inner product  $\langle A|B\rangle = 0$ .

### STEP1: INITIALIZATION:

Now, we define the vector  $|h\rangle$  as an equal superposition of all the inputs by applying the Hadamard gate on  $|000\rangle$  as follows:

$$\begin{aligned} H^{\otimes 3} |000\rangle &= \frac{1}{\sqrt{2^3}} \sum_{x \in \{0,1\}^3} |x\rangle \\ &= \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) \\ |h\rangle &= \sqrt{\frac{2}{8}} \left( \frac{1}{\sqrt{2}}(|011\rangle + |100\rangle) \right) + \sqrt{\frac{6}{8}} \left( \frac{1}{\sqrt{6}}(|000\rangle + |001\rangle + |010\rangle + |101\rangle + |110\rangle + |111\rangle) \right) \\ |h\rangle &= \sqrt{\frac{a}{N}} |A\rangle + \sqrt{\frac{b}{N}} |B\rangle \end{aligned}$$

**STEP2: REFLECTION ABOUT  $|B\rangle$ :**

Now we perform the reflection about  $|B\rangle$  by the action of  $R_{|B\rangle} = Z_f$  on the state  $|h\rangle$  (recall that  $Z_f|x\rangle = (-1)^{f(x)}|x\rangle$ ) which gives:

$$\begin{aligned}|X'\rangle &= Z_f|h\rangle = Z_f \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) \\&= \frac{1}{\sqrt{8}}(Z_f|000\rangle + Z_f|001\rangle + Z_f|010\rangle + Z_f|011\rangle + Z_f|100\rangle + Z_f|101\rangle + Z_f|110\rangle + Z_f|111\rangle) \\&= \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle + |110\rangle + |111\rangle) \\&= -\sqrt{\frac{2}{8}}\left(\frac{1}{\sqrt{2}}(|011\rangle + |100\rangle)\right) + \sqrt{\frac{6}{8}}\left(\frac{1}{\sqrt{6}}(|000\rangle + |001\rangle + |010\rangle + |101\rangle + |110\rangle + |111\rangle)\right) \\|X'\rangle &= -\sqrt{\frac{a}{N}}|A\rangle + \sqrt{\frac{b}{N}}|B\rangle\end{aligned}$$

Thus, we can clearly see that the component in the direction of  $|A\rangle$  are reversed while the components in the direction of  $|B\rangle$  are preserved and hence the resulting  $|X'\rangle$  is a reflection about  $|B\rangle$ .

**STEP3: REFLECTION ABOUT  $|X'\rangle$ :**

Now, we are required to perform a reflection of  $|X'\rangle$  about  $|h\rangle$  by the application of the Diffusion operator. Thus, applying  $-D = -(I - 2|h\rangle\langle h|)$  operator on  $|X'\rangle$  gives

$|X''\rangle$  as:

$$\begin{aligned}
 |X''\rangle &= -D|X'\rangle = -(I - 2|h\rangle\langle h|)|X'\rangle = -(|X'\rangle - 2|h\rangle\langle h|X'\rangle) \\
 &= 2\langle h|X'\rangle|h\rangle - |X'\rangle \\
 &= 2\left(\sqrt{\frac{a}{N}}\langle A| + \sqrt{\frac{b}{N}}\langle B|\right)\left(-\sqrt{\frac{a}{N}}|A\rangle + \sqrt{\frac{b}{N}}|B\rangle\right)|h\rangle - |X'\rangle \\
 &= 2\left(\frac{b}{N} - \frac{a}{N}\right)|h\rangle - |X'\rangle \\
 &= \frac{2(b-a)}{N}\sqrt{\frac{a}{N}}|A\rangle + \frac{2(b-a)}{N}\sqrt{\frac{b}{N}}|B\rangle + \sqrt{\frac{a}{N}}|A\rangle - \sqrt{\frac{b}{N}}|B\rangle \\
 &= \left(\frac{3b-a}{N}\right)\sqrt{\frac{a}{N}}|A\rangle + \left(\frac{b-3a}{N}\right)\sqrt{\frac{b}{N}}|B\rangle \\
 &= \frac{3(6)-2}{8}\sqrt{\frac{2}{8}}|A\rangle + \frac{6-3(2)}{8}\sqrt{\frac{6}{8}}|B\rangle \\
 &= \frac{16}{8}\frac{1}{2}|A\rangle + 0 \\
 &= |A\rangle \\
 &= \frac{1}{\sqrt{2}}(|100\rangle + |011\rangle)
 \end{aligned}$$

Initially the angle between  $|h\rangle$  and  $|B\rangle$  was  $\langle h|B\rangle = \sqrt{\frac{b}{N}} = \cos\theta \implies \theta = \cos^{-1}\sqrt{\frac{6}{8}} = \frac{\pi}{6}$ . After the transformation we can clearly see that the angle between  $|X''\rangle$  and  $|B\rangle$  is  $\langle X''|B\rangle = 0 = \cos\theta \implies \theta = \cos^{-1}0 = \frac{\pi}{2}$ . Thus, the angle changed from  $\frac{\pi}{6}$  to  $\frac{\pi}{2}$ . Thus, we can clearly see that the angle changed from  $\theta$  to  $3\theta$ . Now, we need to repeat this reflection transformation  $k = \lfloor \frac{\pi}{4}\sqrt{8} \rfloor = 2$ . Thus, we need to perform this once again. But note that we have already arrived at  $|A\rangle$  and doing this once again will bring us farther from  $|A\rangle$ . This is the problem with the grover's algorithm. Since, we don't know the number of possible outcomes i.e.  $a = 1$  or  $a \geq 1$ , we assumed that  $a = 1$  and decided to perform the iterations based on  $a = 1$  which comes out as  $k = \lfloor \frac{\pi}{4} \rfloor = 2$ . But, if we perform the operation once again then the angle between  $|X''\rangle$  and  $|B\rangle$  would become  $5\theta = 5\frac{\pi}{6} > \frac{\pi}{2}$ , thus it overshoots, decreasing the probability of outcomes being from state  $|A\rangle$ . In order to improve over this we introduce modified grover's algorithm which states to set  $m=1$ , pick  $k \in 1, \dots, m+1$  uniformly and perform the grover's algorithm  $k$  times and then perform the measurement, and stop only if we get  $x$  such that  $f(x) = 1$ . Else, if

$m < \sqrt{N}$  set  $m = \lfloor (8/7)m \rfloor$ , else fail.

**STEP4: MEASUREMENT:**

Thus, supposed we pick  $k = 1$  and then we perform the Grover's algorithm  $k = 1$  times, we would get the state  $|X''\rangle = |A\rangle$ ,

$$|X''\rangle = |A\rangle = \frac{1}{\sqrt{2}}(|011\rangle + |100\rangle)$$

thus upon performing measurements we would get the required search strings  $x = |100\rangle$  and  $x = |011\rangle$  with the property that  $f(x) = 1$  with equal probability. The Modified Grover's Algorithm is further explained in the following complexity analysis section (refer section 11.4).

## 11.4 Complexity Analysis

Here we answer why the number of times to iterate is  $\lfloor \frac{\pi}{4}\sqrt{N} \rfloor$  and thus the time complexity of the Grover's algorithm is  $\mathcal{O}(\sqrt{N})$ . Recall, that after  $k$  iterations of applying operator  $G$  on  $|X\rangle$ , the state  $|X\rangle$  is:

$$\cos((2k+1)\theta)|B\rangle + \sin((2k+1)\theta)|A\rangle$$

The goal is to measure some element  $x \in A$ , so we would like the state  $|X\rangle$  to be as close to  $|A\rangle$  as possible. In other words, we want the probability of measuring  $|A\rangle$  to be as high as possible which consequently means that the probability of measuring  $|B\rangle$  to be as low as possible. Thus, we want,

$$\sin^2((2k+1)\theta) \approx 1$$

then we require the minimum value of  $k$  with which we can achieve this,

$$(2k+1)\theta \approx \frac{\pi}{2}$$

will suffice, so we should choose

$$k \approx \frac{\pi}{4\theta} - \frac{1}{2}$$

Of course  $k$  must be an integer, which is why we can only hope to approximate the quantity. Thus, putting  $\theta = \sin^{-1} \sqrt{\frac{a}{N}}$ , we get,

$$k \approx \frac{\pi}{4 \sin^{-1} \sqrt{\frac{a}{N}}} - \frac{1}{2}$$

Thus, since  $k$  must be an integer,

$$k = \lfloor \frac{\pi}{4 \sin^{-1} \sqrt{\frac{a}{N}}} \rfloor$$

Note that  $k$  depends on angle  $\theta$  which in turn depends on  $a$  which is the number of solutions. In general, we might not know how many solutions there are so we can't pick the value of  $a$  and thus, don't know the number of iterations  $k$  to perform.

### 11.4.1 Unique Search Problem

This is the case which Lov Grover considered in his original paper in 1996 where he introduced Grover's Algorithm. Suppose for simplicity, which also turns out to be the worst case we would have  $a = 1$  i.e. only one unique solution string  $x$  satisfying such that  $f(x) = 1$  and hence,  $\sin^{-1} \theta \approx \theta \implies \sin^{-1} \frac{1}{\sqrt{N}} \approx \frac{1}{\sqrt{N}}$ .

$$k \approx \frac{\pi}{4} \sqrt{N} - \frac{1}{2}$$

$$k \approx \frac{\pi}{4} \sqrt{N}$$

Thus, we use  $k = \lfloor \frac{\pi}{4} \sqrt{N} \rfloor$  for our algorithm. Now, recall that the operator  $G$  includes the operator  $Z_f$  which queries the black box (oracle)  $B_f$ . Thus, the number of queries needed by the algorithm is equal to the number of times we use the Grover operator which is also  $k$  and thus, the queries required for the algorithm is  $\mathcal{O}(\sqrt{N})$ .

Note that  $a = 1$  represents only a special case which is coincidentally also the worst case for the searching algorithm. Thus, with the choice of  $k = \lfloor \frac{\pi}{4} \sqrt{N} \rfloor$ , the probability of finding the single  $x$  such that  $f(x) = 1$  is

$$\sin^2 \left( \left( 2 \lfloor \frac{\pi}{4} \sqrt{N} \rfloor + 1 \right) \sin^{-1} \frac{1}{\sqrt{N}} \right)$$

Thus, the limit of this probability is 1 as  $N$  goes to infinity, and it is at least  $1/2$ . This is as shown in the following table 11.2

N	Probability
2	0.5000000000
4	1.0000000000
8	0.9453125000
16	0.9613189697
32	0.9991823155
64	0.9965856808
128	0.9956198657
256	0.9999470421
512	0.9994480262
1024	0.9994612447
2048	0.9999968478
4096	0.9999453461
8192	0.9999157752
16384	0.9999997811
32768	0.9999868295
65536	0.9999882596
131072	0.9999992587
262144	0.9999978382
524288	0.9999997279

Table 11.2: Unique Search ( $a = 1$ ): Success Probability at various  $N$ 

So even in the worst case, repeating the algorithm some small constant number of times and evaluating  $f$  at the output each time will find the unique  $x$  such that  $f(x) = 1$  with very high probability.

Notice that these probabilities are not strictly increasing. In particular, we have an interesting anomaly when  $N = 4$ , where we get a solution with certainty. It can be proved analytically that

$$p(N, 1) \geq 1 - \frac{1}{N}$$

where  $p(N, 1)$  is the probability of finding the required search string  $x$  upon measurement. Notice that even a weak bound such as  $p(N, 1) \geq 1/2$  establishes the utility of Grover's algorithm. For whatever measurement outcome  $x$  we obtain from running the procedure, we can always check to see if  $f(x) = 1$  using a single query to  $f$ . And if we fail to obtain the unique string  $x$  for which  $f(x) = 1$  with probability at most  $1/2$  by running the procedure once, then after  $m$  independent runs of the procedure we will have failed to obtain this unique string  $x$  with probability at most  $2^{-m}$ . That is using  $\mathcal{O}(m\sqrt{N})$  queries to  $f$ , we will obtain the unique solution  $x$  with probability

at least  $1 - 2^{-m}$ . Using the better bound  $p(N, 1) \geq 1 - \frac{1}{N}$  reveals that the probability to find  $s \in A$  using this method is actually at least  $1 - N^{-m}$ . Thus, as can be seen that the probabilities for the case  $a = 1$  approaches 1 as we increase  $N$ .

### 11.4.2 Multiple Solutions

Now we discuss the general case. In the case when we do not know that  $a = 1$ , the situation is more challenging. For example, if  $a = 4$  but we still choose  $k = \lfloor \frac{\pi}{4} \sqrt{N} \rfloor$ , the probability of success is given as:

$$\sin^2 \left( \left( 2 \lfloor \frac{\pi}{4} \sqrt{N} \rfloor + 1 \right) \sin^{-1} \sqrt{\frac{a}{N}} \right)$$

thus, the success probability goes to 0 in the limit of large  $N$  as seen in table 11.3.

N	Probability
4	1.0000000000
8	0.5000000000
16	0.2500000000
32	0.0122070313
64	0.0203807689
128	0.0144530758
256	0.0000705058
512	0.0019310741
1024	0.0023009083
2048	0.0000077506
4096	0.0002301502
8192	0.0003439882
16384	0.0000007053
32768	0.0000533810
65536	0.0000472907
131072	0.0000030066
262144	0.0000086824
524288	0.0000010820

Table 11.3: Multiple Solutions: Success Probability at various N

This is because we are effectively rotation twice as far as we should. Generalizing what was claimed earlier, it can be proved that

$$p(N, a) \geq 1 - \frac{a}{N}$$

This lower bound of  $1 - \frac{a}{N}$  on the probability of success is slightly peculiar in that more solutions implies a worse lower bound - but under the assumption that  $a$  is significantly smaller than  $N$ , nevertheless conclude that the probability of success is reasonably high. As before, the mere fact that  $p(N, a)$  is reasonably large implies the algorithm's usefulness. It is also the case that

$$p(N, a) \geq \frac{a}{N}$$

This lower bound describes the probability that a string  $x \in \{0, 1\}^n$  selected uniformly at random is a solution - so Grover's algorithm always does at least as well as random guessing. In fact, Grover's algorithm is random guessing when  $k = 0$ . As the number of elements  $a$  varies, so does the angle  $\theta$ , which can have a significant effect on the algorithm's probability of success. In case we know  $a$  in advance and we choose the corresponding value of  $k$  as

$$k = \lfloor \frac{\pi}{4 \sin^{-1} \sqrt{\frac{a}{N}}} \rfloor$$

then the probability  $p(N, a)$  for finding one of the solutions upon measuring is given by

$$p(N, a) \geq \max\left\{1 - \frac{a}{N}, \frac{a}{N}\right\}$$

thus, as  $a/N$  goes smaller and smaller the probability approaches 1. We now need to see how does the number of queries  $k$  depend on  $N$  and  $a$ .

$$\sin^{-1}(x) \geq x \quad \text{for every } x \in [0, 1]$$

$$\begin{aligned} \theta &= \sin^{-1} \left( \sqrt{\frac{a}{N}} \right) \geq \sqrt{\frac{a}{N}} \\ \implies k &\leq \frac{\pi}{4\theta} \leq \frac{\pi}{4} \sqrt{\frac{N}{a}} \\ k &= \mathcal{O} \left( \sqrt{\frac{N}{a}} \right) \end{aligned}$$

In the case when the number of solutions is unknown i.e. we don't know the value of  $a$ . When we don't know the number of solutions, there are different strategies for dealing with this problem. One possibility is simply choose a random value uniformly for

$$k \in \{1, \dots, \lfloor \frac{\pi}{4} \sqrt{N} \rfloor\}$$

The idea is that we are giving the state vector  $|X\rangle$  a random rotation. This does not make the probability of getting a solution close to 1 but the probability of finding a solution (if one exists) will be at least 40%. We can now boost the probability of getting the correct solution exponentially by repeating this process choosing  $k$  independently each time. The chance of not finding a solution decreases exponentially in the number of times we repeat. For example to increase probability to 99% we can run this algorithm 10 times with  $k$  chosen independently each time. The number of queries is still  $\mathcal{O}(\sqrt{N})$  (or evaluations of  $f$ ) even when we don't know the number of solutions in advance. Still operating under the assumption  $a \geq 1$ , the algorithm will fail to find an  $x$  with  $f(x) = 1$  with probability at most  $3/4$ . Repeating some constant number of times until a "good"  $x$  is found quickly decreases error.

A somewhat better strategy is to apply the method above for successively larger values. Specifically, instead of choosing a random  $k$  in the range  $1, \dots, \lfloor \frac{\pi}{4}\sqrt{N} \rfloor$ , do the following:

---

**Algorithm 2** Modified Grover's Algorithm for  $a \geq 1$ 


---

- 1: Set  $m = 1$ .
  - 2: Choose  $k \in \{1, \dots, m + 1\}$  uniformly at random and run Grover's algorithm for this choice of  $k$ . If the algorithm finds an  $x$  such that  $f(x) = 1$ , then output  $x$  and halt.
  - 3: If  $m > \sqrt{N}$  then "fail". Else set  $m = \lfloor (8/7)m \rfloor$  and goto step 2.
- 

This will succeed in finding  $x \in A$  with probability at least  $1/4$  after  $\mathcal{O}(\sqrt{\frac{N}{a}})$  queries. By repeating step 2 a constant number of times during each iteration of the loop, the error decreases exponentially. (the number  $8/7$  just happens to be a number that works for the analysis, which we will not discuss. The point is that  $m$  increases exponentially, but not too fast to cause the algorithm to fail). Another choice is to use  $\lfloor \frac{5}{4}k \rfloor$ . The rate of increase of  $k$  must be carefully balanced; slower rates require more queries, higher rates decrease success probability. If the number of solutions is  $a \geq 1$ , then the number of queries (or evaluations of  $f$ ) required is  $\mathcal{O}(\sqrt{N/a})$ . If there are no solutions  $\mathcal{O}(\sqrt{N})$  queries are required.

### 11.4.3 Trivial Case

Finally we need to deal with the special cases when  $a = 0$  or  $b = 0$ . Obviously if  $a = 0$ , the output  $x$  of the algorithm will never satisfy  $f(x) = 1$  because there are no such values of  $x$ . It is easy to check directly that Grover's Algorithm will output a choice of  $x \in \{0, 1\}^n$  that is uniformly distributed in this case. The situation where

one of A and B is empty happens when  $f$  is constant; A is empty when  $f(x) = 0$  for every  $x \in \{0, 1\}^n$ , and B is empty when  $f(x) = 1$  for every  $x \in \{0, 1\}^n$ . This means that

$$Z_f |h\rangle = \pm |h\rangle$$

and therefore

$$\begin{aligned} G |h\rangle &= (2 |h\rangle \langle h| - I) Z_f |h\rangle \\ &= \pm (2 |h\rangle \langle h| - I) |h\rangle \\ &= \pm |h\rangle \end{aligned}$$

So, irrespective of the number of iterations  $k$  we perform in these cases, the measurement always reveals a uniform random string  $x \in \{0, 1\}^n$ . Supposing that we do not know  $a$ , if we run Grover's Algorithm as described above and always measure a value of  $x$  for which  $f(x) = 0$ , we can reasonable conclude with high confidence that  $a = 0$ . It is also the case when  $b = 0$  that Grover's Algorithm will output a uniformly distributed choice of  $x$ . Of course in this case  $f(x) = 1$  for all  $x$ , so the problem is trivially solved.

## 11.5 Conclusion

Grover's algorithm is asymptotically optimal within the query model. It means that it is not possible to come with an algorithm for the search problem (unique or multiple solutions) that uses asymptotically fewer than  $\mathcal{O}(\sqrt{n})$  queries in the worst case. This has been proved rigorously and was known even before Grover's algorithm was discovered. This algorithm is broadly applicable as it can be used as a subroutine in other quantum algorithms to use its quadratic speedup on top of other algorithms. The technique used in Grover's algorithm has been generalized and this is called Amplitude Amplification which is then applied to another quantum algorithms to essentially boost its success probability quadratically then it would have been possible classically.

## 11.6 Amplitude Amplification

## 11.7 Applications of Grover's Algorithm

### 11.7.1 Application: Satisfiability

## 11.8 Implementation using Qiskit

### 11.8.1 Background

Usage estimate: 4 seconds (NOTE; This is an estimate only. Your runtime may vary)

Amplitude amplification is a general purpose quantum algorithm, or subroutine, that can be used to obtain a quadratic speedup over a handful of classical algorithms. Grover's algorithm was the first to demonstrate this speedup on unstructured search problems. Formulating a Grover's search problem requires an oracle function that marks one or more computational basis states as the states we are interested in finding, and an amplification circuit that increase the amplitude of marked states, consequently suppressing the remaining states.

Here, we demonstrate how to construct Grover oracles and use the *GroverOperator* from the Qiskit circuit library to easily set up a Grover's search instance. The runtime *Sampler* primitive allows seamless execution of Grover circuits.

### 11.8.2 Requirements

Before starting this tutorial, ensure that you have the following installed:

- Qiskit SDK 1.0 or later, with visualization support ( pip install 'qiskit[visualization]' )
- Qiskit Runtime (pip install qiskit-ibm-runtime) 0.22 or later

### 11.8.3 Setup

Here we import the small number of tools we need for this tutorial.

```
1 # Built-in modules
2 import math
3
4 # Imports from Qiskit
5 from qiskit import QuantumCircuit
```

```

6 from qiskit.circuit.library import GroverOperator, MCMT, ZGate
7 from qiskit.visualization import plot_distribution
8
9 # Imports from Qiskit Runtime
10 from qiskit_ibm_runtime import QiskitRuntimeService
11 from qiskit_ibm_runtime import SamplerV2 as Sampler

1 # To run on hardware, select the backend with the fewest number of jobs
  in the queue
2 service = QiskitRuntimeService(channel="ibm_quantum")
3 backend = service.least_busy(operational=True, simulator=False)
4 backend.name

```

This output one of the ibm quantum computers (say 'ibm\_kyoto').

### 1. Step 1: Map classical inputs to a quantum problem:

Grover's algorithm requires an oracle that specifies one or more marked computational basis states, where "marked" means a state with a phase of -1, A controlled-Z gate, or its multi-controlled generation over N qubits, marks the  $2^N - 1$  state ('1' \* N bit-string). Marking basis states with one or more '0' in the binary representation requires applying X-gates on the corresponding qubits before and after the controlled-Z gate; equivalent to having an open-control on that qubit. In the following code, we define an oracle that does just that, marking one or more input basis states defined through their bit-string representation. The MCMT gate is used to implement the multi-controlled Z gate.

```

1 def grover_oracle(marked_states):
2     """Build a Grover oracle for multiple marked states
3
4     Here we assume all input marked states have the same number of
5     bits
6
7     Parameters:
8         marked_states (str or list): Marked states of oracle
9
10    Returns:
11        QuantumCircuit: Quantum circuit representing Grover oracle
12    """
13    if not isinstance(marked_states, list):
14        marked_states = [marked_states]
15    # Compute the number of qubits in circuit
16    num_qubits = len(marked_states[0])
17    qc = QuantumCircuit(num_qubits)

```

```

18 # Mark each target state in the input list
19 for target in marked_states:
20     # Flip target bit-string to match Qiskit bit-ordering
21     rev_target = target[::-1]
22     # Find the indices of all the '0' elements in bit-string
23     zero_inds = [ind for ind in range(num_qubits) if rev_target.
24     startswith("0", ind)]
25     # Add a multi-controlled Z-gate with pre- and post-applied X-
26     # gates (open-controls)
27     qc.x(zero_inds)
28     qc.compose(MCMT(ZGate(), num_qubits - 1, 1), inplace=True)
29     qc.x(zero_inds)
30 return qc

```

### Specific Grover's Instance

Now that we have the oracle function, we can define a specific instance of Grover search. In this example we will mark two computational state out of the eight available in a three-qubit computational space:

```

1 marked_states = ["011", "100"]
2
3 oracle = grover_oracle(marked_states)
4 oracle.draw(output="mpl", style="iqp")

```

This output is as shown in the following figure 11.5

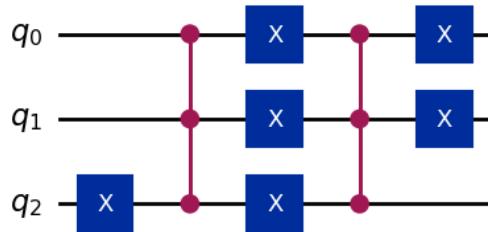


Figure 11.5: Specific Grover's Instance

### GroverOperator

The built-in Qiskit GroverOperator takes an oracle circuit and returns a circuit that is composed of the oracle circuit and a circuit that amplifies the states

marked by the oracle. Here, we decompose the circuit to see the gates within the operator:

```
1 grover_op = GroverOperator(oracle)
2 grover_op.decompose().draw(output="mpl", style="iqp")
```

This outputs as shown in the figure 11.6.

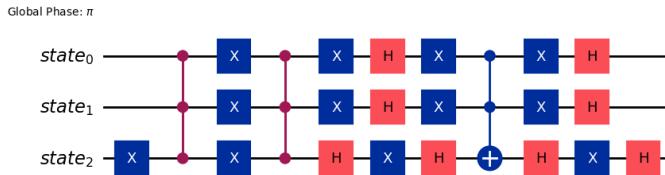


Figure 11.6: Grover Operator

Repeated application of this grover\_op circuit amplify the marked states, making them the most probable bit-strings in the output distribution from the circuit. There is an optimal number of such application that is determined by the ratio of marked states to the total number of possible computational states:

```
1 optimal_num_iterations = math.floor(
2     math.pi / (4 * math.asin(math.sqrt(len(marked_states)) / 2 *
3         grover_op.num_qubits)))
3 )
```

### Full grover circuit

A complete Grover experiment starts with a Hadamard gate on each qubit; creating an even superposition of all computational basis states, followed the Grover operator (grover\_op) repeated the optimal number of times. Here we make use of the QuantumCircuit.power(INT) method to repeatedly apply the Grover operator.

```
1 qc = QuantumCircuit(grover_op.num_qubits)
2 # Create even superposition of all basis states
3 qc.h(range(grover_op.num_qubits))
4 # Apply Grover operator the optimal number of times
5 qc.compose(grover_op.power(optimal_num_iterations), inplace=True)
6 # Measure all qubits
7 qc.measure_all()
8 qc.draw(output="mpl", style="iqp")
```

This output the following figure 11.7

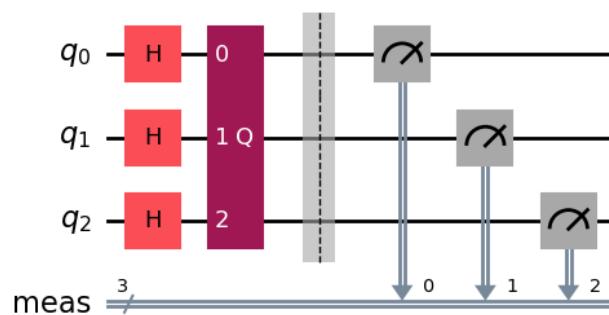


Figure 11.7: Full Grover Circuit

## 2. Step 2: Optimize problem for Quantum Execution

```
1 from qiskit.transpiler.preset_passmanagers import
   generate_preset_pass_manager
2
3 target = backend.target
4 pm = generate_preset_pass_manager(target=target,
   optimization_level=3)
5
6 circuit_isa = pm.run(qc)
7 circuit_isa.draw(output="mpl", idle_wires=False, style="iqp")
```

This outputs the following 11.8.



Figure 11.8: Optimized Circuit

### 3. Step 3: Execute using Qiskit Primitives

Amplitude amplification is a sampling problem that is suitable for execution with the Sampler runtime primitive.

Note that the run() method of Qiskit Runtime SampleV2 takes an iterable of primitive unified blocs (PUBs). For sampler, each PUB is an iterable in the format (circuit, parameter\_values). However, at a minimum, it takes a list of quantum circuit(s).

```

1 # To run on local simulator:
2 #   1. Use the SatetvectorSampler from qiskit.primitives instead
3 sampler = Sampler(backend=backend)
4 sampler.options.default_shots = 10_000
5 result = sampler.run([circuit_isa]).result()
6 dist = result[0].data.meas.get_counts()

```

### 4. Step 4: Post-Process, return result in classical format

```
1 plot_distribution(dist)
```

The result upon measurement is as shown in the figure 11.9

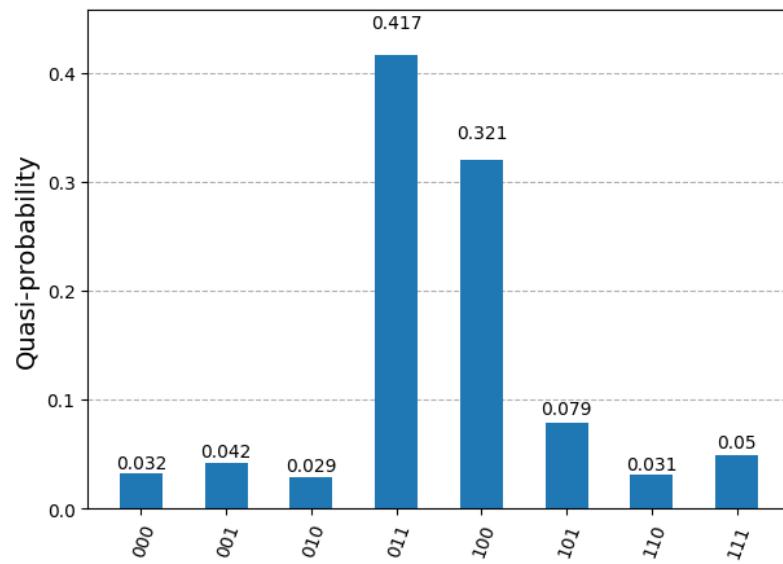


Figure 11.9: Probability plot

# Chapter 12

## Phase Estimation

We know that any classical Boolean circuit can be converted to a reversible (therefore unitary) circuit that efficiently implements the function computed by the original circuit. For example, if the original Boolean circuit computes the Function

$$f : \{0, 1\}^n \rightarrow \{0, 1\}^m$$

then the reversible circuit efficiently implements the transformation

$$U_f : |x\rangle |y\rangle \rightarrow |x\rangle |y \oplus f(x)\rangle$$

for all  $x \in \{0, 1\}^n$  and  $y \in \{0, 1\}^m$ , possibly using some ancilla qubits that we do not mention explicitly. Moreover, if

$$f : \{0, 1\}^n \rightarrow \{0, 1\}^n$$

is invertible and we have a Boolean circuit for efficiently computing both  $f$  and  $f^{-1}$ , then we can construct an efficient reversible circuit that performs the transformation

$$P_f |x\rangle = |f(x)\rangle$$

for all  $x \in \{0, 1\}^n$ .

### 12.1 Phase Estimation

Suppose we are given a Quantum circuit  $Q$  which acts on  $n$  qubits (i.e. takes input as  $n$  qubits and obviously because of reversibility and unitary transformation outputs  $n$  qubits). Thus, corresponding to that Circuit's Unitary transformation we have a

Unitary matrix  $U$  of dimensions  $2^n \times 2^n$ . Now, recall that because  $U$  is unitary, we know that it has a complete, orthonormal collection of eigenvectors

$$|\psi_1\rangle, |\psi_2\rangle, \dots, |\psi_{2^n}\rangle$$

with corresponding eigenvalues (Recall, that the eigen values of a Unitary matrix are complex numbers of unit Modulus (refer Appendix A.1.7) and thus can be written in the form  $e^{2\pi i\theta}$  where  $\theta$  is a real number.)

$$e^{2\pi i\theta_1}, e^{2\pi i\theta_2}, \dots, e^{2\pi i\theta_{2^n}}$$

where  $\theta_1, \theta_2, \dots, \theta_{2^n}$  are real numbers. This means that

$$U|\psi_j\rangle = e^{2\pi i\theta_j}|\psi_j\rangle$$

for all  $j = 1, 2, \dots, 2^n$ . Thus, now we define the **PHASE ESTIMATION PROBLEM** as follows:

#### *PHASE ESTIMATION PROBLEM*

**Problem 12.1.1.** *Input: A quantum circuit  $Q$  that performs a unitary operation  $U$ , along with a quantum state  $|\psi\rangle$  that is promised to be an eigenvector of  $U$ :*

$$U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$$

*Output: An approximation to  $\theta \in [0, 1)$ .*

*Remark.* Note that we have made no specific requirements on the precision to which  $\theta$  must be approximated. It will turn out that for an arbitrary Quantum circuit  $Q$  and eigen vector  $|\psi\rangle$ , the number  $\theta$  can be efficiently approximated by the procedure that we will describe, but only to low precision (to a logarithmic number of bits in the circuit size). However, for certain choices of  $U$  it will be possible to achieve much higher precision, and when we apply our methods to factoring this is the case in which we will be interested.

##### 12.1.1 Classical Method

Using a classical computer, we can estimate  $\theta$  using  $U|\psi\rangle \oslash |\psi\rangle$ , where  $\oslash$  stands for the element-wise division operation. Specifically, if  $|\psi\rangle$  is indeed an eigen vector and  $\langle j|\psi\rangle \neq 0$ , for any  $j$  in the computational basis, then we can extract the phase from

$$\langle j|U|\psi\rangle / \langle j|\psi\rangle = e^{2\pi i\theta}$$

Unfortunately, such an element-wise division operation cannot be efficiently implemented on a quantum computer. In this chapter we look at some of the most basic variants.

### 12.1.2 Hadamard Test

For computing the expectation value of an Unitary operator with respect to a general state (not necessarily an eigen vector state) i.e.  $\langle \psi | U | \psi \rangle$ . Note that  $U$  is Unitary and not necessarily Hermitian, this does not correspond to the measurement of a physical observable. Instead the real and imaginary part of the expectation value need to be measured separately. The (real) Hadamard test is a quantum algorithm that computes the real part of the expectation value of a unitary operator with respect to a state. The circuit is as shown in the figure 12.1

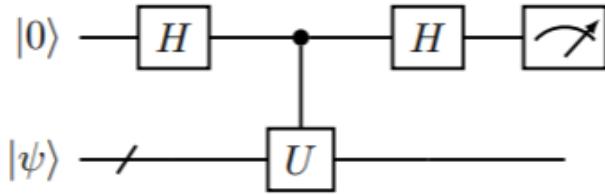


Figure 12.1: Hadamard Test

Note that the circuit transforms  $|0\rangle |\psi\rangle$  as:

$$(H \otimes I)(|0\rangle |\psi\rangle) \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) |\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle |\psi\rangle) + \frac{1}{\sqrt{2}}(|1\rangle |\psi\rangle)$$

Then performing a Controlled Unitary operation:

$$(I \otimes c - U)\left(\frac{1}{\sqrt{2}}(|0\rangle |\psi\rangle) + \frac{1}{\sqrt{2}}(|1\rangle |\psi\rangle)\right) = \frac{1}{\sqrt{2}}(|0\rangle |\psi\rangle + |1\rangle U |\psi\rangle)$$

Now applying the Hadamard transformation on the first qubit we get:

$$\begin{aligned} (H \otimes I)\frac{1}{\sqrt{2}}(|0\rangle |\psi\rangle + |1\rangle U |\psi\rangle) &\rightarrow \frac{1}{\sqrt{2}}(H|0\rangle |\psi\rangle + H|1\rangle U |\psi\rangle) \\ \frac{1}{\sqrt{2}}(H|0\rangle |\psi\rangle + H|1\rangle U |\psi\rangle) &= \frac{1}{2}(|0\rangle (|\psi\rangle + U |\psi\rangle) + |1\rangle (|\psi\rangle - U |\psi\rangle)) \end{aligned}$$

Because the second qubit is not normalized as  $\| |\psi\rangle + U |\psi\rangle \| \neq 1$ , thus we normalize it as shown (Recall that for normalizing we divide by the norm of the vector given

as  $\sqrt{\langle\psi|\psi\rangle}$ ). Thus, here we use  $\sqrt{\langle(I+U^\dagger)(I+U)|\psi\rangle}$  and upon solving for this we get the following:

$$\frac{\sqrt{2+2Re(\langle\psi|U|\psi\rangle)}}{2} \left( |0\rangle \otimes \frac{(|\psi\rangle + U|\psi\rangle)}{\sqrt{2+2Re(\langle\psi|U|\psi\rangle)}} \right) + \\ \frac{\sqrt{2-2Re(\langle\psi|U|\psi\rangle)}}{2} \left( |1\rangle \otimes \frac{(|\psi\rangle - U|\psi\rangle)}{\sqrt{2-2Re(\langle\psi|U|\psi\rangle)}} \right)$$

Now upon measurement of the first qubit, the probability of measuring the first qubit to be 0 is:

$$p(0) = \frac{2+2Re(\langle\psi|U|\psi\rangle)}{4} = \frac{1}{2}(1+Re(\langle\psi|U|\psi\rangle))$$

which is very well defined since  $-1 \leq Re(\langle\psi|U|\psi\rangle) \leq 1$ .

To obtain the imaginary part we can use the circuit as shown in the figure 12.2 called the (imaginary) Hadamard test.

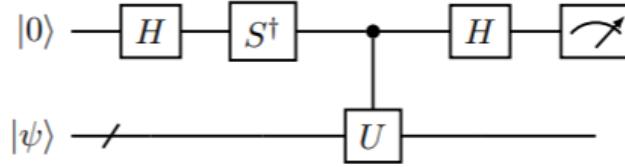


Figure 12.2: Hadamard Test

Similarly doing similar calculations as done above the circuit transforms  $|0\rangle|\psi\rangle$  to the state:

$$\frac{1}{2}|0\rangle(|\psi\rangle\iota U|\psi\rangle) + \frac{1}{2}(|\psi\rangle+\iota U|\psi\rangle)$$

Therefore the probability of measuring the first qubit in the state  $|0\rangle$  is:

$$\frac{1}{2}(1+Im(\langle\psi|U|\psi\rangle))$$

Thus, combining the results of the two circuits, we obtain the estimate to  $\langle\psi|U|\psi\rangle$ .

### Overlap Estimate using Swap Test

it is an application of the Hadamard test called the swap test, which is used to estimate the overlap of two quantum states  $\langle\phi|\psi\rangle$ . The quantum circuit for the swap test is as shown in the figure 12.3.

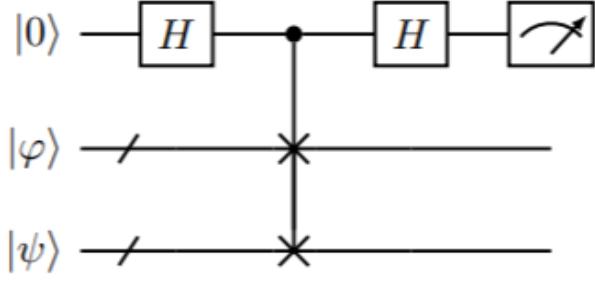


Figure 12.3: Swap Test

Note that it is exactly the Hadamard test with U being the n-swap gate. Using the same calculations as done previously, the state of the system after the first hadamard gate acting on  $|0\rangle|\phi\rangle|\psi\rangle$  is:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|\phi\rangle|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle|\phi\rangle|\psi\rangle + |1\rangle|\phi\rangle|\psi\rangle)$$

Then the controlled swap gate acts on the state as:

$$\frac{1}{\sqrt{2}}(|0\rangle|\phi\rangle|\psi\rangle + |1\rangle|\psi\rangle|\phi\rangle) = \frac{1}{\sqrt{2}}(|0\rangle|\phi\rangle|\psi\rangle + |1\rangle|\psi\rangle|\phi\rangle)$$

Then the second Hadamard gate acts on the first qubit as:

$$\begin{aligned} & \frac{1}{2}(|0\rangle|\phi\rangle|\psi\rangle + |0\rangle|\psi\rangle|\phi\rangle - |1\rangle|\psi\rangle|\phi\rangle + |1\rangle|\phi\rangle|\psi\rangle) \\ &= \frac{\sqrt{2+2|\langle\psi|\phi\rangle|^2}}{2} \left( |0\rangle \otimes \frac{|\phi\rangle|\psi\rangle + |\psi\rangle|\phi\rangle}{\sqrt{2+2|\langle\psi|\phi\rangle|^2}} \right) + \frac{\sqrt{2-2|\langle\psi|\phi\rangle|^2}}{2} \left( |1\rangle \otimes \frac{|\phi\rangle|\psi\rangle - |\psi\rangle|\phi\rangle}{\sqrt{2-2|\langle\psi|\phi\rangle|^2}} \right) \end{aligned}$$

Now, upon measurement of the first qubit, the probability of measuring the first qubit to be 0 is:

$$p(0) = \frac{1}{2}(1 + |\langle\phi|\psi\rangle|^2)$$

### Overlap estimate with Relative phase information

In the swap test, the quantum states  $|\phi\rangle$  and  $|\psi\rangle$  can be black-box states, and in such a scenario obtaining an estimate to  $|\langle\phi|\psi\rangle|$  is the best one can do. In order to

retrieve the relative phase information and to obtain  $\langle \phi | \psi \rangle$ , we need to have access to the unitary circuit preparing  $|\phi\rangle$  and  $|\psi\rangle$ .

$$U_\phi |0^n\rangle = |\phi\rangle, \quad U_\psi |0^n\rangle = |\psi\rangle$$

Then we have  $\langle \phi | \psi \rangle = \langle 0^n | U_\phi^\dagger U_\psi | 0^n \rangle$ . Now putting  $U = U_\phi U_\psi$  in the (real) and (imaginary) Hadamard test and putting the state  $|0^n\rangle$  in the second register, thus calculating the expectation value, we can obtain the real and imaginary parts of  $\langle \phi | \psi \rangle$ .

### Single Phase Estimation

The hadamard test can be also be used to derive the simplest version of the phase estimation based on success probabilities. Apply the Hadamard test as shown in the figure 12.1. with  $U, \psi$  satisfying the equation  $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$ . Then the probability of measuring the first qubit to be in state  $|1\rangle$  is:

$$p(1) = \frac{1}{2}(1 - Re(e^{2\pi i\theta})) = \frac{1}{2}(1 - \cos(2\pi\theta))$$

Therefore,

$$\theta = \pm \frac{\arccos(1 - 2p(1))}{2\pi} \pmod{1}$$

In order to quantify the efficiency of this procedure recall that if  $p(1)$  is far away from 0 (or close to 1) i.e.  $(2\theta \bmod 1)$  is far away from 0, in order to approximate  $p(1)$  (and hence  $\theta$ ) to additive precision  $\epsilon$ , the number of samples need is  $\mathcal{O}(1/\epsilon^2)$  using the formula dervied from statistics (If we know that the count  $X$  of "successes" in a group of  $n$  observations with sucess probability  $p$  has a binomial distribution with mean  $np$  and variance  $np(1-p)$ , then we are able to derive information about the distribution of the sample proportion, the count of successes  $X$  divided by the number of observations  $n$ . By the multiplicative properties of the mean, the mean of the distribution of  $X/n$  is equal to the mean of  $X$  divided by  $n$ , or  $np/n = p$ . This proves that the sample proportion is an unbiased estimator of the population proportion  $p$ . The variance of  $X/n$  is equal to the variance of  $X$  divided by  $n^2$ , or  $(np(1-p))/n^2 = (p(1-p))/n$ . This formula indicates that as the size of the sample increases, the variance decreases.):

$$N \geq \frac{p(1-p)}{\epsilon^2}$$

Now assume that  $\theta$  is very close to 0 and we would like to estimate  $\theta$  to additive precision  $\epsilon$ . Note that

$$p(1) \approx (2\pi\theta)^2 = \mathcal{O}(\epsilon^2)$$

Then  $p(1)$  needs to be estimated to precision  $\mathcal{O}(\epsilon^2)$ , and again the number of samples needed is  $\mathcal{O}(1/\epsilon^2)$ . The case when  $\theta$  is very close to  $1/2$  or  $1$  is similar.

Note that the circuit in figure 12.1 cannot distinguish the sign of  $\theta$  (or whether  $\theta \geq 1/2$  when restricted to the interval  $[0, 1]$ ). To do this, we can use the circuit in figure 12.2 by replacing  $S^\dagger$  with  $S$ , so that the success probability of measuring 1 in the computational basis is

$$p(1) = \frac{1}{2}(1 + \sin(2\pi\theta))$$

This gives

$$\theta = \begin{cases} \in [0, 1/2], & p(1) \geq \frac{1}{2} \\ \in (1/2, 1), & p(1) < \frac{1}{2} \end{cases}$$

Unlike previous estimate, in order to correctly estimate the sign, we only require  $\mathcal{O}(1)$  accuracy, and run figure 12.2 for a constant number of times (unless  $\theta$  very close to 0 or  $\pi$ ).

### 12.1.3 Kitaev's Method - for Quantum Phase Estimation

The number of measurements needed to estimate  $\theta$  to precision  $\epsilon$  is  $\mathcal{O}(1/\epsilon^2)$ . A quadratic improvement in precision (i.e.  $\mathcal{O}(1/\epsilon)$ ) can be achieved by means of the quantum phase estimation using Kitaev's method. Assuming that the eigen value can be represented using  $d$  bits i.e.,

$$\theta = (\theta_{d-1} \dots \theta_0)$$

In the simplest scenario, we assume  $d = 1$ , and  $\theta = \theta_0, \theta_0 \in \{0, 1\}$ . Thus, if  $\theta_0 = 0$  then  $e^{2\pi i 0} = e^{\pi i 0} = 1$  and if  $\theta_0 = 1$  then  $e^{2\pi i 0.5} = e^{\pi i}$ . Then  $e^{2\pi i \theta} = e^{\pi i \theta_0}$ . Performing the real Hadamard test, we have  $p(1) = \frac{1}{2}(1 - \text{Re}(e^{\pi i \theta_0})) = \frac{1}{2}(1 - \cos(\pi \theta_0))$ .

Now,  $p(1) = 0$  if  $\theta_0 = 0$  and  $p(1) = 1$  if  $\theta_0 = 1$ . In either case the result is deterministic, and one measurement is sufficient to determine the value of  $\theta_0$ .

Next, consider  $\theta = 0.0 \dots 0 \theta_0_{dbits}$ . To determine the value of  $\theta_0$  we need to reach precision of  $\epsilon < 2^{-d}$ . The method in Single Phase qubit estimation requires  $\mathcal{O}(1/\epsilon^2) = \mathcal{O}(2^{2d})$  repeated measurements or number of queries to  $U$ . The observation from Kitaev's method is that if we can have access to  $U^j$  for a suitable power of

$j$ , then the number of queries to  $U$  can be reduced. More specifically, if we can query  $U^{2^{d-1}}$ , then the circuit in figure 12.4 with  $j = d - 1$  gives

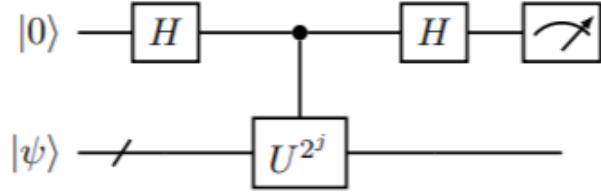


Figure 12.4: Kitaev's Method

$$p(1) = \frac{1}{2}(1 - \text{Re}(\langle \psi | U^{2^{d-1}} | \psi \rangle)) = \frac{1}{2}(1 - \text{Re}(e^{2\pi i 2^{d-1}\theta}))$$

Now the value of  $2^{d-1}0..0\theta_0 = \theta_0/2 = 0.\theta_0$  (in binary):

$$p(1) = \frac{1}{2}(1 - \cos(2\pi \cdot \theta_0)) = \begin{cases} 0, & \theta_0 = 0 \\ 1, & \theta_0 = 1 \end{cases}$$

This is the basic idea behind Kitaev's method: use a more complex circuit (and in particular, with a larger circuit depth) to reduce the total number of queries. As a general strategy, instead of estimating  $\theta$  from a single number, we assume access to  $U^{2^j}$ , and estimate  $\theta$  bit-by-bit.

$$2^j\theta = \theta_{d-1} \dots \theta_{d-j} \cdot \theta_{d-j-1} \dots \theta_0 = .\theta_{d-j-1} \dots \theta_0 \pmod{1}$$

Note that now upon running the circuit we would get  $p(1)$  thus we can estimate  $.\theta_{d-j-1} \dots \theta_0 \pmod{1}$ . But, for values such as 0.0111 and 0.1000 we need to be careful in order to distinguish, in case of bit-by-bit estimation, and the two numbers can be arbitrarily close to each other. Here, we first describe the algorithm and then analyze its performance. The algorithm works for any  $\theta$  and then the goal is to estimate its  $d$  bits. For simplicity of the analysis we assume that  $\theta$  is exactly represented by  $d$  bits.

Now applying circuit in figure 12.4 for  $j = 0, 1, \dots, d - 3$  and the corresponding circuit to determine its sign. For each  $j$  we can estimate  $p(0)$ , so that the error in  $2^j\theta$  is less than  $1/16$  for all  $j$ . (This can happen with a sufficient high success probability. For simplicity, we assume that this happens with certainty). The measured result is denoted by  $\alpha_j$ . This means that any perturbation must

be due to the 5th digit in binary representation. For example, if  $2^j\theta = 0.11100$ , then  $\alpha_j = 0.11011$  is an acceptable result with an error of  $0.00001 = 1/32$ , but  $\alpha_j = 0.11110$  is not acceptable since the error is  $0.0001 = 1/16$ . We then round  $\alpha_j \pmod{1}$  by its closest 3-bit estimate denoted by  $\beta_j$ , i.e.  $\beta_j$  is taken from the set  $\{0.000, 0.001, 0.010, 0.011, 0.100, 0.101, 0.110, 0.111\}$ .

Consider the example of  $2^j\theta = 0.11110$ , if  $\alpha_j = 0.11101$  then  $\beta_j = 0.111$ . But, if  $\alpha_j = 0.11111$ , then  $\beta_j = 0.000$  is not required to approximate. Another example is  $2^j\theta = 0.11101$ , if  $\alpha_j = 0.11110$  then both  $\beta_j = 0.111$  (rounded down) and  $\beta_j = 0.000$  (rounded up) are acceptable. We can pick one of them at random. We will show later that the uncertainty in  $\alpha_j, \beta_j$  is not detrimental to the success of the algorithm.

Second, we perform post-processing. Start from  $j = d - 3$ , we can estimate  $\theta_2\theta_1\theta_0$  to accuracy  $1/16$ , which reconvers these three bits exactly. The values of these three bits will be taken from  $\beta_{d-3}$  directly. Then we proceed with the iteration for  $j = d - 4, \dots, 0$ , we assign

$$\theta_{d-j-1} = \begin{cases} 0, & |.0\theta_{d-j-2}\theta_{d-j-3} - \beta_j| < 1/4 \\ 1, & |.1\theta_{d-j-2}\theta_{d-j-3} - \beta_j| > 1/4 \end{cases}$$

#### 12.1.4 Phase Estimation Procedure

Suppose that  $U$  is a unitary transformation acting on  $n$  qubits (recall that dimensions of  $U$  will be  $2^n \times 2^n$ ), and suppose  $m$  is any positive integer. Then we can define another unique unitary transformation  $\Lambda_m(U)$  acting on  $n + m$  qubits (thus of dimesnions  $2^{m+n} \times 2^{m+n}$ ) as follows:

$$\Lambda_m(U) |k\rangle |\phi\rangle = |k\rangle (U^k |\phi\rangle)$$

for all choices of  $k \in \{0, \dots, 2^m - 1\}$ . Here,  $|k\rangle$  will be of dimensions  $2^m$  and  $|\phi\rangle$  will be of dimensions  $2^n$ . Thus,  $|k\rangle |\phi\rangle$  will be of dimensions  $2^m 2^n = 2^{m+n}$  and thus the matrix  $\Lambda_m(U)$  of dimensions  $2^{m+n} \times 2^{m+n}$  acting on  $|k\rangle |\phi\rangle$ . In other words, the first  $m$  qubits specify the number of times that  $U$  is to be applied to the remaining  $n$  qubits. This can be seen in the following circuit figure 12.5:

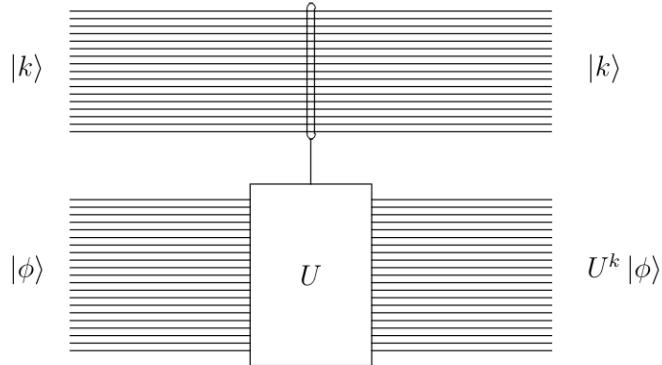


Figure 12.5: circuit

### Important Note

Note that it is possible that  $n$  and  $m$  differ significantly. Now, if given a Quantum circuit  $Q$  that performs the Unitary operation  $U$ , there is no guarantee that you will be able to build a quantum circuit that efficiently implements  $\Lambda_m(U)$  for your choice of  $m$ . For example, if  $m \approx n$ , you would probably require that  $U$  has some special properties that allow an efficient implementation. This is because the number of times ( $k$ ) that  $U$  may effectively need to be performed can be exponential in  $m$  (since  $k = \mathcal{O}(2^m)$ ). If  $m = \mathcal{O}(\log n)$ , you can always construct an efficient implementation of  $\Lambda_m(U)$  (assuming that you have a circuit  $Q$  that efficiently implements  $U$ ).

We are not concerned with the details of how one would construct  $\Lambda_m(U)$  for small values of  $m$  given a circuit  $Q$  implementing  $U$ , because our interest will be focused on a particular choice of  $U$  where it is easy to implement  $\Lambda_m(U)$ , even for  $m = \Theta(n)$ . Specifically, the transformation  $U$  will correspond to modular multiplication by a fixed number  $a$ , and so  $\Lambda_m(U)$  will correspond to modular exponentiation, which we have already shown is efficiently implementable. Let us forget about the specifics and suppose that we have a quantum circuit implementing  $\Lambda_m(U)$  for some particular choice of  $a$  (which may be large or small).

Consider the following circuit diagram 12.6:

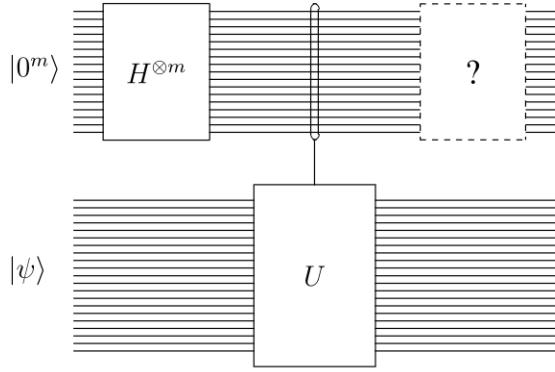


Figure 12.6: circuit

Note that the input to the second collection of qubits is the state  $|\psi\rangle$ , which is promised to be an eigenvector of  $U$ . Now consider the state after the  $\Lambda_m(U)$  operation is performed. The initial state is  $|0^m\rangle |\psi\rangle$ , and after the hadamard transformations are performed the state becomes:

$$\frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} |k\rangle |\psi\rangle$$

Then the  $\Lambda_m(U)$  transformation is performed, which transforms the state to

$$\frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} |k\rangle |U^k |\psi\rangle\rangle$$

Now, let us use the fact that  $|\psi\rangle$  is an eigenvector of  $U$  to simplify this expression. Specifically we assume that

$$U |\psi\rangle = e^{2\pi i \theta} |\psi\rangle$$

for some real number  $\theta \in [0, 1]$ , which is the value that we are trying to approximate. Applying  $U^k$  to  $|\psi\rangle$  is equivalent to applying  $U$  to  $|\psi\rangle$  a total of  $k$  times, so

$$U^k |\psi\rangle = (e^{2\pi i \theta})^k |\psi\rangle = e^{2\pi i k \theta} |\psi\rangle$$

Thus, the state after the  $\Lambda_m(U)$  operation is performed is

$$\frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} |k\rangle e^{2\pi i k \theta} |\psi\rangle = \frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i k \theta} |k\rangle |\psi\rangle$$

This is called **Phase kickback**. The first  $m$  qubits and the last  $n$  qubits are uncorrelated at this point, given that they are in tensor product state:

$$\left( \frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i k \theta} |k\rangle \right) \otimes |\psi\rangle$$

So, if we discard the last  $n$  qubits we are left with the state:

$$\frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i k \theta} |k\rangle$$

**Example 12.1.1. A simple case:** Recall that our goal is to approximate  $\theta$ . Suppose that  $\theta$  happens to have a special form:

$$\theta = \frac{j}{2^m}$$

for some integer  $j \in \{0, \dots, 2^m - 1\}$ . (This makes  $\theta$  take only certain distinct values thus, in general, it is not correct to assume that  $\theta$  has this form, but it is helpful to consider this case first.) Then the above statement can be rewritten as:

$$\frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i k \theta} |k\rangle = \frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i k j / 2^m} |k\rangle$$

Now, let  $\omega = e^{2\pi i / 2^m}$ , then the above expression can be written as:

$$\frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} \omega^{kj} |k\rangle$$

Now let us define:

$$|\phi_j\rangle = \frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} \omega^{kj} |k\rangle$$

for each  $j \in \{0, \dots, 2^m - 1\}$ . we know that the state of the first  $m$  qubits of our circuit is one of the states  $\{|\phi_j\rangle : j = 0, \dots, 2^m - 1\}$  and the goal is to determine which one. **Recall that given a known set of states, an unknown state from that collection is given to you, and your goal is to determine which of the possible states it is. It is possible to solve the problem perfectly if the set**

**of states is orthonormal.** Once we know  $j$ , we know  $\theta$  as well (because we are still considering the special case  $\theta = j/2^m$ ). We have

$$\langle \phi_j | \phi_{j'} \rangle = \frac{1}{2^m} \sum_{k=0}^{2^m-1} \omega^{k(j-j')} = \frac{1}{2^m} \sum_{k=0}^{2^m-1} (\omega^{j-j'})^k$$

Note that when we take the inner product in the above equation only the terms for which  $|k\rangle$  is same survive since the  $|k\rangle$  forms an orthonormal basis. Thus,  $\langle k|k' \rangle = \delta_{kk'}$ . Also, for  $\langle \phi_j |$  recall that because of the conjugate transpose operation the  $\omega$  will be conjugated and thus  $\omega^* = (e^{2\pi i \theta j / 2^m})^* = e^{-2\pi i \theta j / 2^m} = \omega^{-j}$ . Using the formula for geometric series for the term  $\omega^{j-j'}$  we get:

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

Thus, we get:

$$\langle \phi_j | \phi_{j'} \rangle = \frac{1}{2^m} \frac{\omega^{(j-j')2^m} - 1}{\omega^{j-j'} - 1}$$

Now, since  $\omega^{(2^m)l} = 1$  for any integer  $l = j - j'$  (subtraction of two integers is an integer) we get:

$$\langle \phi_j | \phi_{j'} \rangle = \begin{cases} 1, & \text{if } j = j' \\ 0, & \text{if } j \neq j' \end{cases}$$

Thus, the set  $\{|\phi_0\rangle, \dots, |\phi_{2^m-1}\rangle\}$  is indeed orthonormal. Recall that we transformed from the basis  $\{|0\rangle, \dots, |2^m-1\rangle\}$  to the basis  $\{|\phi_j\rangle : j = 0, \dots, 2^m-1\}$ . Now both the basis are orthonormal basis. Thus, this is a rotation in complex plane and thus a corresponding Unitary matrix exists. There is therefore a Unitary transformation  $F$  that satisfies

$$F |j\rangle = |\phi_j\rangle$$

for all  $j \in \{0, \dots, 2^m-1\}$ . Thus, the transformation  $F$  is the one that performs the rotation in the complex plane. We can describe this matrix explicitly by allowing the vecotrs  $|\phi_j\rangle$  to determine the columns of  $D$ :

$$F = \begin{pmatrix} |\phi_0\rangle & |\phi_1\rangle & \dots & |\phi_{2^m-1}\rangle \end{pmatrix}$$

$$F = \frac{1}{\sqrt{2^m}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{2^m-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(2^m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{2^m-1} & \omega^{2(2^m-1)} & \dots & \omega^{(2^m-1)(2^m-1)} \end{pmatrix}$$

This is the matrix that defines a linear transformation in the complex plane that rotates the basis  $\{|0\rangle, \dots, |2^m - 1\rangle\}$  to the basis  $\{|\phi_0\rangle, \dots, |\phi_{2^m-1}\rangle\}$  (Note that the matrix is not Hermitian but symmetric. Also, it is Unitary since the columns are orthonormal, thus,  $QFT QFT^\dagger = QFT^\dagger QFT = I \implies QFT^\dagger = QFT^{-1}$ ).  $QFT^\dagger = QFT^{-1}$  since the matrix is also Unitary). This is called the *Discrete Fourier Transform*. In the context of Quantum Computing we call it Quantum Fourier Transform, and for that reason it is also sometimes denoted as  $QFT_{2^m}$ . Thus, for convenience we can write it explicitly as:

$$QFT_{2^m} |j\rangle = \frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i j k / 2^m} |k\rangle = |\phi_j\rangle$$

Thus, plugging the inverse of this transformation into our circuit from before, we obtain (using  $QFT^\dagger = QFT^{-1}$ ):

$$QFT_{2^m}^\dagger |\phi_j\rangle = |j\rangle$$

The circuit for this is as shown in the figure 12.7

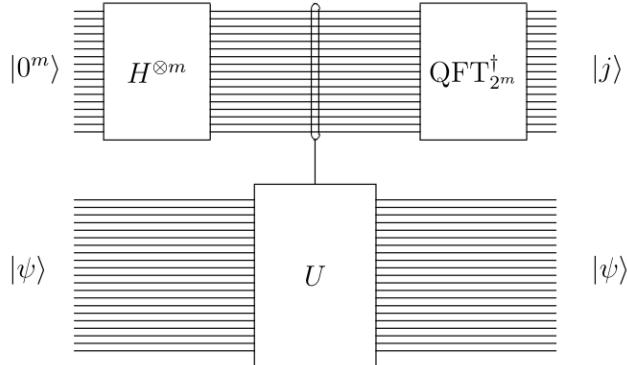


Figure 12.7: circuit

Thus, measuring the first  $m$  qubits and dividing by  $2^m$  tell us precisely the value of  $\theta$ . **But this assumes that:**

$$U |\psi\rangle = e^{2\pi i j / 2^m} |\psi\rangle$$

We still need to prove this for the more general case that  $\theta$  does not have the form  $j/2^m$  for some integer  $j$ . The exact same circuit is used to deal with the general case, but the analysis will become more complicated (**and the answer will only be an approximation to  $\theta$** ). The two major issues to deal with are:

1. What can be said about the measurement in the case that  $\theta$  does not have the form  $j/2^m$  for some integer  $j$ , and
2. Can the Quantum Fourier transform be implemented efficiently?

### For the general values of $\theta$

From the above analysis we know that for an arbitrary value of  $\theta$ , the state of the above circuit immediately before the inverse of the QFT is applied is

$$\frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i k \theta} |k\rangle |\psi\rangle$$

Ignoring the second collection of qubits since it is uncorrelated with the first  $m$  qubits, so we are free to disregard these qubits and consider just the first  $m$  qubits. Applying the transformation  $QFT_{2^m}^\dagger$  to these qubits results in the state:

$$\frac{1}{2^m} \sum_{k=0}^{2^m-1} e^{2\pi i k \theta} \sum_{j=0}^{2^m-1} e^{-2\pi i j k / 2^m} |j\rangle$$

Note that in the above equation we have expanded the  $QFT^\dagger |k\rangle$  in terms of the basis  $\{|j\rangle\}$ . Thus, upon simplifying we get:

$$\frac{1}{2^m} \sum_{k=0}^{2^m-1} \sum_{j=0}^{2^m-1} e^{2\pi i (k\theta - jk/2^m)} |j\rangle$$

Now rearranging and grouping the terms with respect to  $j$  we can rearrange the summations to get:

$$\begin{aligned} & \sum_{j=0}^{2^m-1} \sum_{k=0}^{2^m-1} \frac{1}{2^m} e^{2\pi i (k\theta - jk/2^m)} |j\rangle \\ & \sum_{j=0}^{2^m-1} \left( \sum_{k=0}^{2^m-1} \frac{1}{2^m} e^{2\pi i (k\theta - jk/2^m)} \right) |j\rangle \end{aligned}$$

Thus, upon measurement the probability that the measurement result in outcome  $j$  is therefore.

$$p_j = \left| \frac{1}{2^m} \sum_{k=0}^{2^m-1} e^{2\pi i k (\theta - j/2^m)} \right|^2$$

for each  $j \in \{0, \dots, 2^m - 1\}$ . Again using the summation of geometric series formula to evaluate the above expression we get:

$$p_j = \left| \frac{1}{2^m} \frac{e^{2\pi i 2^m(\theta-j/2^m)} - 1}{e^{2\pi i (\theta-j/2^m)} - 1} \right|^2 = \frac{1}{2^{2m}} \left| \frac{e^{2\pi i (2^m\theta-j)} - 1}{e^{2\pi i (\theta-j/2^m)} - 1} \right|^2$$

Now we are required to show that the probability  $p_j$  is large for values of  $j$  that satisfy  $j/2^m \approx \theta$  and small otherwise.

Recall that all of this is with respect to the eigen vector state  $|\psi\rangle$  and we have assumed that the corresponding eigen value associated with the eigen vector of the Unitary matrix  $U$ ,  $|\psi\rangle$  is  $e^{2\pi i \theta}$ . We are required to approximate  $\theta \in [0, 1)$ . We have shown that it works perfectly for  $\theta = j/2^m$  for some integer  $j \in \{0, \dots, 2^m - 1\}$ , where upon measurement the results in outcome  $j$  with probability 1. Now, we are required to show that the probability  $p_j$  is large for value of  $j$  that satisfy  $j/2^m \approx \theta$  and small otherwise. Thus, when  $\theta$  does not have the form  $j/2^m$  for some integer  $j$ . We had determined that the probability associated with each possible outcome  $j \in \{0, \dots, 2^m - 1\}$  of the measurement is:

$$p_j = \frac{1}{2^{2m}} \left| \frac{e^{2\pi i (2^m\theta-j)} - 1}{e^{2\pi i (\theta-j/2^m)} - 1} \right|^2$$

Because, we already dealt with the case that  $\theta = j/2^m$  for some integer  $j$ , We assume now that  $\theta \neq j/2^m$ . Now to prove that the probability  $p_j$  is highest when  $\theta \approx j/2^m$ , we assume that  $\theta = j/2^m + \epsilon$ . Thus,

$$e^{2\pi i \theta} = e^{2\pi i (j/2^m + \epsilon)}$$

for some real number  $\epsilon$  with  $|\epsilon| \leq \frac{1}{2^{m+1}}$ , where equality means that the fractional parts of the two sides of the equations agree. This is equivalent to saying:

$$\theta = \frac{j}{2^m} + \epsilon \bmod 1$$

Now we prove a lowerbound on  $p_j$  as follows: Let

$$\begin{aligned} a &= |e^{2\pi i (2^m\theta-j)} - 1| = |e^{2\pi i \epsilon 2^m} - 1| \\ b &= |e^{2\pi i (\theta-j/2^m)} - 1| = |e^{2\pi i \epsilon} - 1| \end{aligned}$$

Thus, we can write the probability  $p_j$  as (using the properties of the modulus of a complex numbers):

$$p_j = \frac{1}{2^{2m}} \left| \frac{a}{b} \right|^2 = \frac{1}{2^{2m}} \frac{a^2}{b^2}$$

Now, to get a lower bound on  $p_j$  we need to get a lower bound on  $a$  and an upper bound on  $b$ . To get a lower bound on  $a$  consider the following figure 12.8:

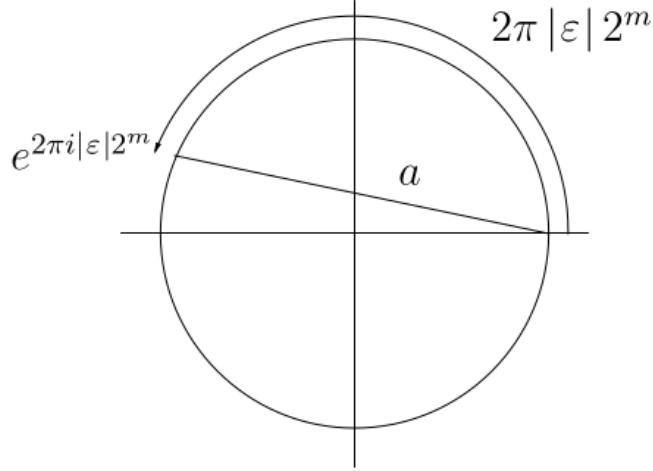


Figure 12.8: Lower bound of a

Now from the figure recall that the ratio of the minor arc length to the chord length can be written as

$$\frac{R\theta}{2R \sin \frac{\theta}{2}} = \frac{\theta}{2 \sin \frac{\theta}{2}} \leq \frac{\pi}{2}$$

with equality at  $\theta = \pi$ . Thus, from the figure 12.8 we can say that:  $\frac{2\pi|\epsilon|2^m}{a} \leq \frac{\pi}{2} \implies a \geq 4|\epsilon|2^m$ . Along the similar lines, we may consider  $b$  along with the fact that the ratio of the arc length to chord length is at least 1:

$$\frac{\theta}{2 \sin \frac{\theta}{2}} = \frac{1}{\frac{\sin \frac{\theta}{2}}{\frac{\theta}{2}}} \geq 1$$

Thus, as  $\theta \rightarrow 0$  the ratio  $\frac{\theta}{2 \sin \frac{\theta}{2}}$  approaches 1. Thus, we can say that  $b \geq 1$ . Thus we can say that the ratio is always greater than 1. Thus,

$$\frac{2\pi|\epsilon|}{b} \geq 1 \implies b \leq 2\pi|\epsilon|$$

Thus, we can say that the probability  $p_j$  is lower bounded by:

$$p_j = \frac{1}{2^{2m}} \frac{a^2}{b^2} \geq \frac{1}{2^{2m}} \frac{(4|\epsilon|2^m)^2}{(2\pi|\epsilon|)^2} = \frac{1}{2^{2m}} \frac{16|\epsilon|^2 2^{2m}}{4\pi^2|\epsilon|^2} = \frac{4}{\pi^2} = 0.405284$$

Thus, the probability is atleast 40.5% of measuring the value of  $j$  when  $\theta$  is close to  $j/2^m$ . This is the probability that every single one of the bits you measure is correct, so that your approximation to  $\theta$  is good to  $m$  bits of precision.

Similarly we can use the above analysis to put upper bounds on the probability of obtaining inaccurate results. Suppose now that for a given value of  $j$  we have

$$e^{2\pi i \theta} = e^{2\pi i(j/2^m + \epsilon)}$$

for some real number  $\epsilon$  with  $\frac{\alpha}{2^m} \leq |\epsilon| < 1/2$ . Here  $\alpha$  is an arbitrary positive number that we can choose later. Thus, we have

$$p_j = \frac{1}{2^{2m}} \frac{a^2}{b^2}$$

for

$$\begin{aligned} a &= |e^{2\pi i(2^m\theta - j)} - 1| = |e^{2\pi i\epsilon 2^m} - 1| \\ b &= |e^{2\pi i(\theta - j/2^m)} - 1| = |e^{2\pi i\epsilon} - 1| \end{aligned}$$

Now, we can put upper bounds on  $a$  and lower bounds on  $b$ . From the figure 12.8, we can say that  $a \leq 2$  and  $b \geq 4|\epsilon|$ . Now we have,

$$p_j \leq \frac{4}{2^{2m}(4|\epsilon|^2)} = \frac{1}{4\alpha^2}$$

This implies that highly inaccurate results are very unlikely. For example, if we consider  $\alpha = 1$ , meaning that our assumption is only that  $|\epsilon| \geq 2^{-m}$ , the probability of obtaining the corresponding value of  $j$  is atmost  $1/4$ . For worse approximations, implying a larger bound on  $|\epsilon|$ , the probability of obtaining the corresponding value of  $j$  quickly becomes very small.

Thus, to have a better result than a  $4/\pi^2$  probability of obtaining an approximation of  $\theta$  upto  $k$  bits of precision. Set  $k = m + 2$  bits, run the phase estimation procedure several times, and to look for the most commonly appearing outcome. At least one outcomes, which is accurate to  $k + 2$  bits of precision, occurs with probability at least  $4/\pi^2$ . Outcome with fewer than  $k$  bits of precision are much less likely as argued above. If you now take the most commonly occurring outcome and round it to  $k$  bits of precision, the probability of correctness approaches 1 exponentially fast in the number of times the procedure is repeated. Notice also that you do not need multiple copies of the state  $|\psi\rangle$  to perform this process, because the state  $|\psi\rangle$  remains on the lower collection of qubits each time the procedure is performed and can be simply fed into the next iteration.

### 12.1.5 Quantum Fourier Transform

Quantum fourier transform is a quantum implementation of Discrete Fourier Transform. Fourier transform has applications in signal processing, linear algebra, and many more. In short, **Fourier Analysis is a tool to describe the internal frequencies of a function.** Here, shown is the **Quantum Algorithm for computing the discrete fourier transform which is exponentially faster than the famous Fast Fourier Transform (FFT) algorithm fo classical computers.**

But, it has certain problems related to measurement. The QFT algorithm to transform the n qubit state vector  $|\alpha\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle + \dots + \alpha_{2^n-1}|2^n-1\rangle$  to the state vector  $|\beta\rangle = \beta_0|0\rangle + \beta_1|1\rangle + \dots + \beta_{2^n-1}|2^n-1\rangle$ , a measurment of  $|\beta\rangle$  will only return one of its n components, and we are not able to recover all the information of the Fourier transform. For this reason, we describe the algorithm as **quantum fourier sampling**.

The Quantum Fourier transform is the generalization of the Hadmard transform. it is very similar, with the exception that QFT introduces phase./ The specific kinds of phases introduced are what we call primitive roots of unity,  $\omega$ . To know more about the primitive roots of unity (refer Appendix A.3.8).

#### Quantum Fourier Transform with quantum gates

The strength of the FFT is that we are able to use the symmetry of the discrete Fourier transform to our advantage. The circuit application of QFt uses the same principle, but because of the power os superposition QFT is even faster.

The QFT is motivated by the FFT so we will follow the same steps, but because this is a quantum algorithm the implmentation fo the steps will be different. We first take the Fourier transform of the odd and even parts, then multiply the odd terms by the phase  $\omega^j$ . Consider the figure 12.9.

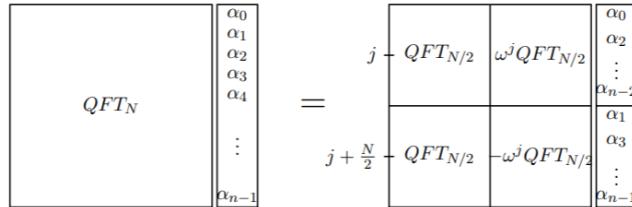


Figure 12.9: circuit

In quantum algorithm, the first step is that the odd and even terms are together

in superposition: the odd terms are those whose least significant bit is 1, and even with 0. Therefore, we can apply  $QFT_{N/2}$  to both the odd and even terms together. We do this by simply applying the  $QFT_{N/2}$  to the  $m-1$  most significant bits, and recombine the odd and even appropriately by applying the Hadmard to the least significant bit as shown in the figure 12.10.

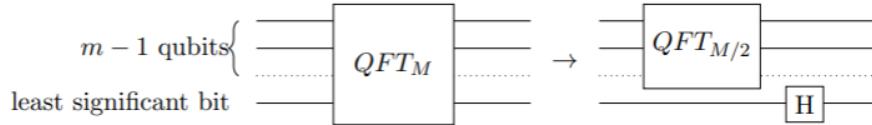


Figure 12.10: circuit

Now to carry out the phase multiplication, we need to multiply each odd term  $j$  by the phase  $\omega_j$ . But remember, an odd number in binary ends with a 1, while an even ends with 0. Thus, we can use the controlled phase shift, where the least significant bits is the control, to multiply only the odd terms by the phase without doing anything to the even terms. Recall that the controlled phase shift is similar to the CNOT gate in that it only applies a phase to the target if the control bit is one.

The phase associated with each controlled phase shift should be equal to  $\omega^j$  where  $j$  is associated to the  $k$ th bit by  $j = 2^k$ . Thus, apply the controlled phase shift to each of the first  $m-1$  qubits , with the least significant bit as the control. With the controlled phase shift and the Hadmard transform,  $QFT_N$  has been reduced to  $QFT_{N/2}$  as shown in the figure 12.11.

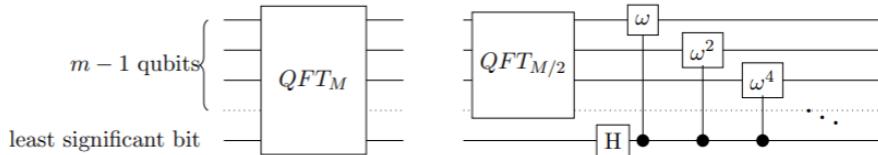


Figure 12.11: circuit

$QFT_N$  is reduced to  $QFT_{N/2}$  and N additional gates

**Example 12.1.2.** Let us contrsuct  $QFT_8$ . Following the algorithm, we will turn  $QFT_8$  into  $QFT_4$  and a few quantum gates. Then continuing on this way we turn  $QFT_4$  into  $QFT_2$  (which is just a Hadamard gate) and another few gates. Controlled phase gates will be represented by  $R_\phi$ .

number of gates necessary to carry out

$QFT_N$  is exactly  $\sum_{i=1}^{\log N} i = \log N(\log N + 1)/2 = \mathcal{O}(\log^2 N)$ . Thus, the circuit is as shown in the figure 12.12.

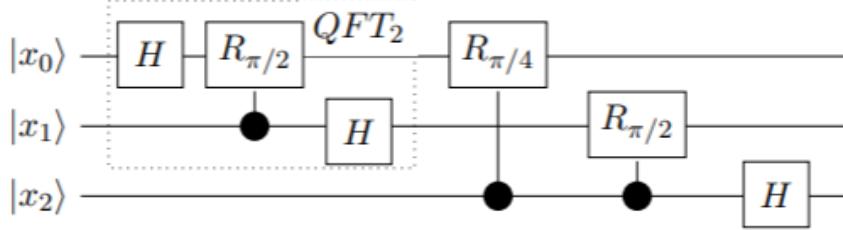


Figure 12.12: circuit

How can quantum Fourier transform may be implemented by quantum circuits. Recall that

$$QFT_{2^m} |j\rangle = \frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i j k / 2^m} |k\rangle$$

Let the notations be ( $N = 2^m$ ):

$$\omega_N = e^{2\pi i / N}$$

for any positive integer  $N$ . Let us also define a unitary mapping  $\tilde{QFT}_{2^m}$  to be the same as  $QFT_{2^m}$  except with the output bits in reverse order. Specifically, if an integer  $k \in \{0, \dots, 2^m - 1\}$  is written in binary notation as  $k_{m-1}, k_{m-2} \dots k_0$  then we define

$$\tilde{QFT}_{2^m} |j_{m-1} j_{m-2} \dots j_0\rangle = \frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} \omega_{2^m}^{jk} |k_0 k_1 \dots k_{m-1}\rangle$$

If we can come up with an efficient implementation of  $\tilde{QFT}_{2^m}$  follows - just reverse the order of the output qubits after performing  $\tilde{QFT}_{2^m}$ . The reason why we consider  $\tilde{QFT}_{2^m}$  rather than  $QFT_{2^m}$  is simply for convenience.

Our description of quantum circuits for performing  $\tilde{QFT}_{2^m}$  for any given value of  $m$  is essentially recursive. Starting with base case  $m=1$ . The Transformation is just  $\tilde{QFT}_2$  is just a fancy name for a Hadamard transformation:

$$\tilde{QFT}_2 |j\rangle = \frac{1}{\sqrt{2}} \sum_{k=0}^1 \omega_2^{jk} |k\rangle = \frac{1}{\sqrt{2}} (|0\rangle + \omega_2^j |1\rangle) = \frac{1}{\sqrt{2}} (|0\rangle + (-1)^j |1\rangle)$$

For general case  $m \geq 2$ , the following circuit computes  $\tilde{QFT}_{2^m}$ :

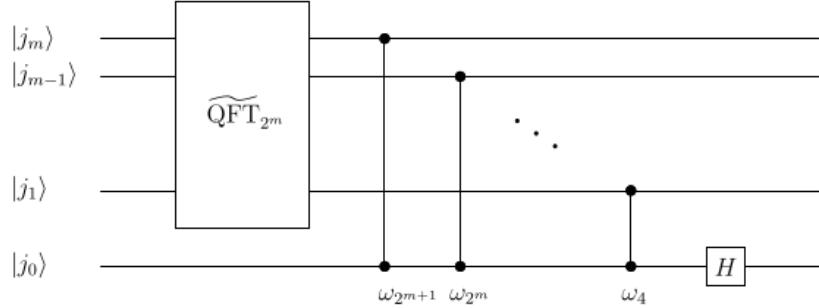


Figure 12.13: QFT circuit

This diagram assume you know how to implement the transformation  $\tilde{QFT}_{2^m}$ , but usuing the fact that  $\tilde{QFT}_2$  is the same as Hadamrd transform we can easily unwind the recursion if we want an explicit description of a circuit.

To show that the circuit works correctly, it suffices as usual to show that it works correctly on classical states. We wish to show that

$$\tilde{QFT}_{2^m} |j_m j_{m-1} j_{m-2} \dots j_0\rangle = \frac{1}{\sqrt{2^{m+1}}} \sum_{k=0}^{2^{m+1}-1} \omega_{2^m}^{jk} |k_0 k_1 \dots k_{m-1} k_m\rangle$$

for each  $j \in \{0, \dots, 2^{m+1} - 1\}$ . Let us write

$$j' = j_m j_{m-1} \dots j_1 = \lfloor j/2 \rfloor$$

$$k' = k_{m-1} k_{m-2} \dots k_0 = k - k_m 2^m$$

# Chapter 13

## Shor's Algorithm

**Definition 13.0.1.** Given a number  $N$  which is a composite number. We are required to find the prime factors of the number  $N$ .

# Chapter 14

## The HHL Algorithm

### 14.1 Introduction

The Harrow-Hassidim-Lloyd (HHL) [?] algorithm for solving large systems of linear equations. Linear system of equations arise in almost any real-life applications from Physics, Maths and Engineering to Financial and Biological Sciences. Typical examples of real-life problems involving solving system of linear equations computationally include, solutions of Partial Differential Equations, the calibration of financial models, fluid simulation of numerical field calculation, optimization, machine learning etc. Most real-life problems generally boils down to a set of Linear Equations. We even approximate non-linear terms in non-linear equations, to linear terms in order to approximate the non-linear equations to linear equations in few cases.

### 14.2 The Linear-System Problem (LSP)

The classical linear-system problem is defined as follows:

**Definition 14.2.1.** Given a matrix  $A \in \mathbb{C}^{N \times N}$  and a vector  $\vec{b} \in \mathbb{C}^N$ , find  $\vec{x} \in \mathbb{C}^N$  satisfying  $A\vec{x} = \vec{b}$ .

In most of the applications it sufficies to find  $\vec{x}$  such that  $\|\vec{x} - \vec{x}_0\| < \epsilon$  for some  $\epsilon > 0$ , where  $\vec{x}_0$  is the exact solution. Assuming that  $A$  is invertible (or  $\text{rank}(A) = N$  i.e. full rank) for simplicity, the solution to the above problem is given by  $\vec{x} = A^{-1}\vec{b}$ . In case,  $A$  is not invertible, we can use the Moore-Penrose Pseudo-Inverse of  $A$  to get the solution which is  $\vec{x} = (A^\dagger A)^{-1} A^\dagger \vec{b}$ , where  $(A^\dagger A)^{-1} A^\dagger$  is called the Moore-Penrose Psuedo-Inverse of  $A$ .

A system of linear equations is called  **$s$ -sparse if  $A$  has at most  $s$  non-zero entries per row or column**. Solving an  $s$ -sparse system of size  $N$  with a classical computer requires  $O(Ns\kappa \log(1/\epsilon))$  time complexity using the conjugate gradient method [?]. Here,  $\kappa$  is the condition number of the system and  $\epsilon$  is the accuracy of the approximation. Overall, the solution on a classical computer is simply  $x = A^{-1}b$ . Note that here the solution  $x$  is not necessarily a normalized vector, and hence cannot be stored as a quantum state. Thus, for solving the problem on a Quantum Computer we define the corresponding QLSP which is the quantum equivalent of the classical LSP.

The Quantum Linear-System Problem (QLSP) is defined as follows:

**Definition 14.2.2.** Given a matrix  $A \in \mathbb{C}^{N \times N}$  and  $|b\rangle \in \mathbb{C}^N$ , find  $|\tilde{x}\rangle \in \mathbb{C}^N$  such that  $\| |x\rangle - |\tilde{x}\rangle \| \leq \epsilon$  for some  $\epsilon > 0$  where  $|x\rangle$  is the exact solution satisfying  $A|x\rangle = |b\rangle$ .

For simplicity, we assume  $|b\rangle$  is normalized and can be efficiently implemented using a transformation  $U_b$  (has a efficient quantum circuit) as a  $n_b$  qubit quantum state. Here, the goal is to produce an n-qubit state whose amplitude-vector (up to normalization and up to  $\epsilon$  error) is a solution to the linear system. Thus,

$$\| |x\rangle - |\tilde{x}\rangle \| \leq \epsilon, \quad |x\rangle = \frac{A^{-1}|b\rangle}{\|A^{-1}|b\rangle\|}$$

We are required to find the normalization constant  $\|A^{-1}|b\rangle\|$  separately. Note that here  $|\tilde{x}\rangle$  will be in a superposition state, thus in order to estimate the amplitude/-coefficients it is required to do a lot of experiments.

The HHL algorithm requires  $A$  to be Hermitian. We can create any given general matrix  $A$  to a Hermitian Matrix using the following method called **Dilation method**.

Without the loss of generality we can assume that  $A$  (through this we can convert the general matrix  $A$  given in classical LSP to a Hermitian matrix for QLSP) is Hermitian since any matrix  $A$  can be made Hermitian using the following method. We define  $A'$  as:

$$A' = \begin{bmatrix} 0 & A \\ A^\dagger & 0 \end{bmatrix} = |1\rangle\langle 0| \otimes A + |0\rangle\langle 1| \otimes A^\dagger$$

and thus, the problem now can be defined as:

$$\begin{bmatrix} 0 & A \\ A^\dagger & 0 \end{bmatrix} \begin{bmatrix} |0\rangle \\ |x\rangle \end{bmatrix} = \begin{bmatrix} |b\rangle \\ |0\rangle \end{bmatrix}$$

thus preserving the original problem  $A|x\rangle = |b\rangle$ . Thus,  $A'|x\rangle' = |b\rangle'$  where  $|x\rangle' = \begin{bmatrix} |0\rangle \\ |x\rangle \end{bmatrix}$ ,  $|b\rangle' = \begin{bmatrix} |b\rangle \\ |0\rangle \end{bmatrix}$  and  $A'$  is Hermitian ( $A'^\dagger = A'$ ). Thus for a given problem in classical LSP as  $A|x\rangle = |b\rangle$ , we can convert it to a QLSP problem as  $A'|x\rangle' = |b\rangle'$  where  $A'$  is Hermitian using the above method. Thus, without the loss of generality for the application of HHL algorithm we can assume that the matrix  $A$  is Hermitian.

The HHL algorithm estimates a function of the solution with time complexity of  $O(\log(N)s^2\kappa^2/\epsilon)$ . The **assumptions/restrictions** on the Linear system in order to make the HHL algorithm work are:

1. **Assume that  $b$  is normalized:** (hence can be stored in a quantum state) We have a unitary matrix  $U_b$  which can prepare  $|b\rangle$  as an  $n$ -qubit quantum state  $|b\rangle = \frac{1}{\|b\|} \sum_i b_i |i\rangle$  using a circuit of  $B$  2-qubit gates. We also assume for simplicity that  $\|b\| = 1$ . Thus, it is possible that using some work registers  $|0\rangle_{n_b}$  to transform it to  $|b\rangle$  using the transformation as shown:

$$|0^{n_b}\rangle \xrightarrow{U_b} |b\rangle_{n_b}$$

2. The matrix  $A$  is  $s$ -sparse and we have sparse access to it. Such sparsity is not essential to the algorithm, and could be replaced by other properties that enable an efficient block-encoding of  $A$ .
3. The matrix  $A$  is well-conditioned: ( $A$  is invertible, thus  $\kappa$  is some finite value) The ratio between its largest and smallest singular value is at most some  $\kappa$ . For simplicity, assume the smallest singular value is  $\geq 1/\kappa$  while the largest is  $\leq 1$ . In other words, all eigenvalues of  $A$  lie in the interval  $[-1, -1/\kappa] \cup [1/\kappa, 1]$ . The smaller the condition number  $\kappa$  is, the better it will be for the algorithm. Let's assume our algorithm knows  $\kappa$ , or at least knows a reasonable upper bound on  $\kappa$ .

## 14.3 The HHL Algorithm

### 14.3.1 Some Mathematical Preliminaries

To encode the problem on a quantum computer we need to rescale the system, by assuming  $b$  and  $x$  to be normalized and map them to the respective quantum states

$|b\rangle$  and  $|x\rangle$ . The  $i$ th component of  $|b\rangle$  corresponds to the amplitude of the  $i$ th basis state of the quantum state  $|b\rangle$ . Thus, we now focus on the rescaled problem

$$A|x\rangle = |b\rangle$$

Since  $A$  is Hermitian ( $A^\dagger = A$ ) thus, a Normal Matrix ( $AA^\dagger = A^\dagger A$ ), hence Unitarily diagonalizable as:

$$A = U\Lambda U^\dagger$$

where  $U$  is a unitary matrix made up of the eigenvectors of  $A$  and  $\Lambda$  is a diagonal matrix with the eigenvalues of  $A$  on the diagonal. Its spectral decomposition can be written in outer product form as:

$$A = \sum_{j=0}^{N-1} \lambda_j |\lambda_j\rangle \langle \lambda_j|, \quad \lambda_j \in \mathbb{R}$$

where  $|\lambda_j\rangle$  are the eigenvectors of  $A$  and  $\lambda_j$  are the corresponding eigenvalues. Then,

$$\begin{aligned} A^{-1} &= (U\Lambda U^\dagger)^{-1} \\ &= U\Lambda^{-1}U^\dagger \end{aligned}$$

Thus spectral decomposition of  $A^{-1}$  is:

$$A^{-1} = \sum_{j=0}^{N-1} \lambda_j^{-1} |\lambda_j\rangle \langle \lambda_j|$$

Now, since  $A$  is unitarily diagonalizable thus, it forms a complete basis set of orthonormal eigenvectors  $\{|\lambda_j\rangle\}$  which span the  $\mathbb{C}^N$  space. Thus, any vector  $|v\rangle$  can be written as a linear combination of the eigenvectors of  $A$ . Thus, we write the right hand side of the system  $|b\rangle$  as:

$$|b\rangle = \sum_{j=0}^{N-1} b_j |\lambda_j\rangle, \quad b_j \in \mathbb{C}$$

The goal of the HHL algorithm is to exit with the readout register in the state

$$\begin{aligned} |x\rangle &= A^{-1}|b\rangle = \left( \sum_{j=0}^{N-1} \lambda_j^{-1} |\lambda_j\rangle \langle \lambda_j| \right) \left( \sum_{k=0}^{N-1} b_k |\lambda_k\rangle \right) \\ &= \left( \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} b_k \lambda_j^{-1} |\lambda_j\rangle \langle \lambda_j| \lambda_k \rangle \right) \end{aligned}$$

Now  $\langle \lambda_j | \lambda_k \rangle = \delta_{jk}$  (because  $A$  is a Hermitian matrix, thus Unitarily diagonalizable and hence has orthonormal eigenvectors), thus the above expression simplifies to:

$$|x\rangle = \sum_{j=0}^{N-1} b_j \lambda_j^{-1} |\lambda_j\rangle$$

which is the required solution to the linear system  $A|x\rangle = |b\rangle$ .

### 14.3.2 Algorithm

Consider the following circuit as shown in the figure 14.1. The first qubit show is initialised to  $|0\rangle$  which is also called the signal qubit. The algorithm uses three quantum register, all set to  $|0\rangle$  at the beginning of the algorithm. One register, which we will denote with the subindex  $n_l$ , is used to store a binary representation of the eigen values of  $A$ . A second register, denoted by  $n_b$ , contains the vector solution. Note that  $N = 2^{n_b}$ . There is an extra register, which we will denote with  $n_a$  for auxillary qubits, used for intermediate steps in the computation. We can ignore any auxillary in the following description as they are  $|0\rangle$  at the beginning of each computation, and are restored back to  $|0\rangle$  at the end of each individual operation.

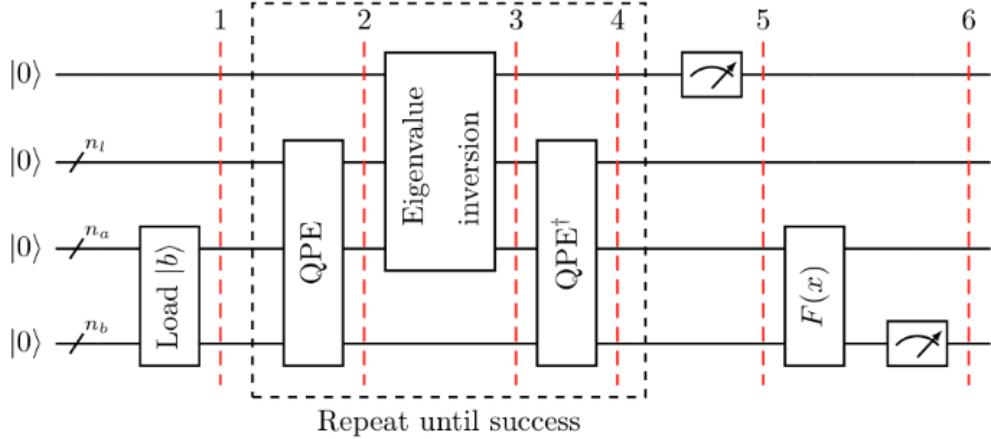


Figure 14.1: HHL Circuit

#### Analysis:

Let us now analyze the working of the above circuit (note that we ignore the ancilla registers throughout the analysis):

1. Load the data  $|b\rangle \in \mathbb{C}$ . That is, perform the transformation

$$|0\rangle |0\rangle_{n_l} |0\rangle_{n_b} \xrightarrow{I \otimes I \otimes U_b} |0\rangle |0\rangle_{n_l} |b\rangle_{n_b}$$

2. Now we are required to apply QPE. Ignoring the Ancilla bits, recall that in QPE given a Unitary matrix  $U$  and one of its eigen vector  $|\psi\rangle$ , we can find the eigenvalue  $e^{\imath 2\pi\theta}$  corresponding to  $|\psi\rangle$  (or specifically approximate  $\theta$  i.e.  $\tilde{\theta}$  upto d bits of accuracy) by applying the QPE algorithm, where  $\theta \approx \tilde{\theta} \in [0, 1]$ . Thus, we have the following transformation:

$$|0^d\rangle |\psi\rangle \xrightarrow{U_{QPE}} |\tilde{\theta}\rangle |\psi\rangle$$

where  $|\tilde{\theta}\rangle$  is a d-bit (in binary) basis state in the computational basis corresponding to the closest binary representation of the actual phase  $\theta \in [0, 1)$  that corresponds to the eigen vector  $|\psi\rangle$ . Thus, first we require a Unitary matrix, here we are required to form a Unitary matrix using A which is a Hermitian matrix. Let us define  $U = e^{\imath 2\pi At}$  where t is some scalar prescaling factor and U is a Unitary matrix (exponential of a Hermitian matrix results in a unitary matrix). Now,

$$e^{\imath 2\pi At} = I + \imath 2\pi At - \frac{1}{2}(2\pi)^2 A^2 t^2 - \frac{\imath}{6}(2\pi)^3 A^3 t^3 + \dots$$

We know that  $A = Q\Lambda Q^\dagger$ , since A is unitarily diagonalizable for some orthonormal matrix Q, upon substituting and simplifying, we get,  $e^{\imath 2\pi At} = Q e^{\imath 2\pi \Lambda t} Q^\dagger$ . Upon writing this in outer product/spectral decomposition form we get,

$$U = e^{\imath 2\pi At} = \sum_{j=0}^{N-1} e^{\imath 2\pi \lambda_j t} |\lambda_j\rangle \langle \lambda_j|$$

Thus, U and A share the same eigen vectors but the eigen values of At are  $2\pi\lambda_j t$  and the eigen values of U are  $e^{\imath 2\pi \lambda_j t}$ , thus we can apply QPE to find the eigen values of A. Also recall that since A is Hermitian matrix, thus, the eigen values of A i.e.  $\lambda_j \in \mathbb{R} \forall j$ . Using QPE, we can find approximate values of the phases and thus, we can write,  $e^{\imath 2\pi \tilde{\theta}_j} \approx e^{\imath 2\pi \lambda_j t} \implies \tilde{\theta}_j \approx \lambda_j t$  (recall that  $\tilde{\theta}_j \in [0, 1)$  which is an approximation to the exact phase  $\theta_j$ ), Upon rearranging the terms we get,

$$\lambda_j \approx \frac{\tilde{\theta}_j}{t}$$

for each  $j \in \{0, \dots, N - 1\}$ . Recall, that here  $t$  is not known to us, thus we need to estimate it. We use this prescaling factor in order to map the eigen values of  $A$  to the range  $[0, 1)$ . There are various methods to estimate  $t$  so that the Eigenvalues of  $A$  are in the range  $[0, 1)$ . Without getting involved on how to find  $t$ , we assume that we know the value of  $t$  so that the eigen values of  $A$  are in  $[0, 1)$ . Let this new eigen values be  $\tilde{\theta}_j \approx \lambda'_j = \lambda_j t$  such that  $0 \leq \lambda'_0 \leq \lambda'_1 \leq \lambda'_2 \leq \dots \leq \lambda'_{N-1} < 1$ . Thus,

$$\tilde{\theta}_j \approx \lambda'_j \quad \forall \quad j \in \{0, \dots, N - 1\}$$

Thus, the phases are approximations to the shifted eigenvalues to be in range  $[0, 1)$ . Here, the QPE problem is as shown in the figure 14.2.



Figure 14.2: QPE

Note that the input to the QPE is not an eigen state but as shown in the figure 14.2, it is a superposition of the eigen states of  $A$ , i.e.  $|b\rangle = \sum_{j=0}^{N-1} b_j |\lambda_j\rangle$ . Thus, the resulting output will be a superposition of  $\tilde{\theta}_j$  i.e.  $\sum_{j=0}^{N-1} b_j |\tilde{\theta}_j\rangle |\lambda_j\rangle$ . Here,  $|\tilde{\theta}_j\rangle$  is one of the basis states in the computational basis corresponding to the closest binary representation of the actual phase  $\theta_j = \lambda'_j = \lambda_j t$  that corresponds to the eigen vector  $|\lambda_j\rangle$ . In other words, say the actual phase corresponding to a eigen state  $|\lambda_j\rangle$  is  $\theta_j \in [0, 1)$  Thus,  $\theta_j \approx 0.\theta_{j_{n_l-1}}\theta_{j_{n_l-2}}\dots\theta_{j_0} = \tilde{\theta}_j$  is the  $n_l$  bit binary approximation of  $\theta_j$ . Thus, the output of the QPE will be a superposition of the  $n_l$  bit binary approximations of the actual phases  $\theta_j$ . At the end of this transformation we get,

$$|0\rangle |0\rangle_{n_l} |0\rangle_{n_b} \xrightarrow{I \otimes QPE} |0\rangle \sum_{j=0}^{N-1} b_j |\tilde{\theta}_j\rangle |\lambda_j\rangle$$

where  $\tilde{\theta}_j$  are binary approximations of the eigen values of  $A$  in the range  $[0, 1)$  corresponding to the eigen vector  $|\lambda_j\rangle$ .

**3. Implementation of the controlled rotations:** Now we are suppose to inverse the eigen values in order to achieve the aim of getting

$$\sum_{j=0}^{N-1} \lambda_j^{-1} b_j |\tilde{\theta}_j\rangle |\lambda_j\rangle$$

from

$$|0\rangle \sum_{j=0}^{N-1} b_j |\tilde{\theta}_j\rangle |\lambda_j\rangle$$

In order to do this we are required to implement controlled rotations. Let us take a slight diversion from the algorithm to learn about the implementation of controlled rotation in order to implement it efficiently for the algorithm. Consider the following proposition.

**Proposition 14.3.1.** (*Controlled rotation given rotation angles*). *Let  $0 \leq \theta < 1$  has exact  $d$ -bit fixed point representation  $\theta = 0.\theta_{d-1}\dots\theta_0$  be its  $d$ -bit fixed point representation. Then there is a  $(d+1)$ qubit unitary  $U_\theta$  such that*

$$U_\theta : |0\rangle |\theta\rangle \rightarrow (\cos(\pi\theta) |0\rangle + \sin(\pi\theta) |1\rangle) |0\rangle$$

*Proof.* Recall that  $R_Y$  gate is as follows:

$$R_\theta^Y = R_Y(\theta) = e^{-i\frac{\theta}{2}\sigma_Y} = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$$

Now putting  $\theta = 2\tau$  we get

$$R_Y(2\tau) = e^{i\tau\sigma_Y} = \begin{bmatrix} \cos(\tau) & -\sin(\tau) \\ \sin(\tau) & \cos(\tau) \end{bmatrix}$$

Here  $R_Y(\cdot)$  performs a single-qubit rotation around the y-axis. For any  $j \in \{0, \dots, 2^d - 1\}$  with its binary representation  $j = j_{d-1} \dots j_0$ , we have

$$\frac{j}{2^d} = (0.j_{d-1} \dots j_0)$$

So we choose  $\tau = \pi(0.j_{d-1} \dots j_0)$ , which can be further simplified putting  $j_k = \theta_k \forall k \in \{d-1, \dots, 0\}$ . Thus, we define

$$\begin{aligned} U_\theta &= \sum_{j=0}^{N-1} e^{-i\pi(0.j_{d-1} \dots j_0)\sigma_Y} \otimes |j\rangle \langle j| \\ &= \sum_{\theta_{d-1}, \dots, \theta_0} e^{-i\pi(0.\theta_{d-1} \dots \theta_0)\sigma_Y} \otimes |\theta_{d-1} \dots \theta_0\rangle \langle \theta_{d-1} \dots \theta_0| \end{aligned}$$

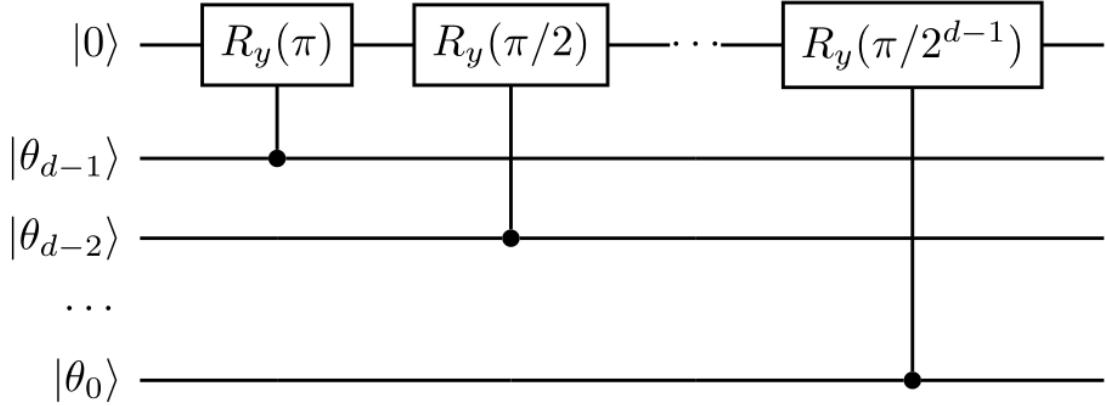


Figure 14.3: Controlled Rotation

$$\begin{aligned}
 U_\theta &= \sum_{\theta_{d-1}, \dots, \theta_0} e^{-i\pi\frac{\theta_{d-1}}{2}\sigma_Y} e^{-i\pi\frac{\theta_{d-2}}{4}\sigma_Y} \dots e^{-i\pi\frac{\theta_0}{2^d}\sigma_Y} \otimes |\theta_{d-1} \dots \theta_0\rangle \langle \theta_{d-1} \dots \theta_0| \\
 &= \sum_{\theta_{d-1}, \dots, \theta_0} R_Y(\pi\theta_{d-1}) R_Y(\pi\frac{\theta_{d-2}}{2}) \dots R_Y(\pi\frac{\theta_0}{2^{d-1}}) \otimes |\theta_{d-1} \dots \theta_0\rangle \langle \theta_{d-1} \dots \theta_0| \\
 &= \sum_{\theta_{d-1}, \dots, \theta_0} R_Y(\pi\theta_{d-1}) R_Y(\frac{\pi}{2}\theta_{d-2}) \dots R_Y(\frac{\pi}{2^{d-1}}\theta_0) \otimes |\theta_{d-1} \dots \theta_0\rangle \langle \theta_{d-1} \dots \theta_0|
 \end{aligned}$$

Thus, applying  $U_\theta$  to  $|0\rangle |\theta\rangle$  gives the desired result since corresponding to a particular value of  $\theta$  only one of the terms from the summation with which its inner product results in 1 will survive (since all the basis states are orthonormal). The resulting quantum circuit for the controlled rotation is as shown in the figure 14.3.  $\square$

Thus, as shown in the circuit in figure 14.3 based upon the input binary state of  $\theta_{d-1} \dots \theta_0$  the corresponding  $R_Y$  gates will be acted on the signal bit (which was initialised to  $|0\rangle$ ) where each  $\theta_k$  will act as the control bit for the corresponding  $R_Y(\pi/2^{d-k-1})$ .

In other words, the controlled rotation implementation will perform a transformation as:

$$U_\phi |0\rangle |\phi\rangle \rightarrow (\cos(\pi\phi) |0\rangle + \sin(\pi\phi) |1\rangle) |\phi\rangle$$

where  $\phi \in [0, 1]$ . This is a sequence of single-qubit rotations on the signal qubit, each controlled by a single qubit.

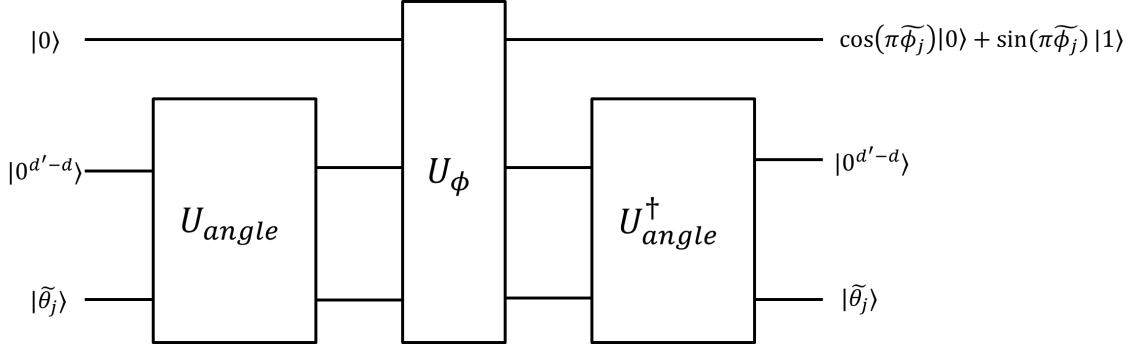


Figure 14.4: Entire Controlled Rotation

Back to the problem, we are required to inverse the eigen values which can be done using implementation of controlled rotation operation. Say we define

$$\phi_j = \frac{1}{\pi} \sin^{-1}(C/\tilde{\theta}_j)$$

where  $C > 0$  be a lower bound to  $\lambda'_0$ , so that  $0 < C/\tilde{\theta}_j < 1$  for all  $j$ . Recall that  $\tilde{\theta}_j \approx \lambda'_j$  and  $\tilde{\theta}_j \in [0, 1]$ . Writing all of this in  $d'$  bit binary representation we get,

$$\tilde{\phi}_j = \frac{1}{\pi} \sin^{-1}(C/\tilde{\theta}'_j) \approx \phi_j = \frac{1}{\pi} \sin^{-1}(C/\tilde{\theta}_j)$$

For simplicity, we assume that  $d'$  is large enough so that the error of the fixed point representation is negligible in this step. The mapping

$$U_{angle} |0^{d'-d}\rangle |\tilde{\theta}_j\rangle = |\tilde{\phi}_j\rangle$$

can be implemented using *classical arithmetics circuits*, which may require  $\text{poly}(d')$  gates and an additional working register of  $\text{poly}(d')$  qubits, which is not displayed here. Therefore the entire controlled rotation operation needed for the HHL algorithm is as given by the circuit 14.4. Therefore through the uncomputation  $U_{angle}^\dagger$ , the  $d'-d$  ancilla qubits also become a working register. Discard the working register we obtain a unitary  $U_{CR}$  satisfying

$$U_{CR} |0\rangle |\tilde{\theta}_j\rangle = (\cos(\pi\tilde{\phi}_j) |0\rangle + \sin(\pi\tilde{\phi}_j) |1\rangle) |\tilde{\theta}_j\rangle = \left( \sqrt{1 - \frac{C^2}{\tilde{\theta}'_j^2}} |0\rangle + \frac{C}{\tilde{\theta}'_j} |1\rangle \right) |\tilde{\theta}_j\rangle$$

This is the unitary circuit used in the HHL algorithm. Now recall the state of

the system after applying the QPE was

$$|0\rangle \sum_{j=0}^{N-1} b_j |\tilde{\theta}_j\rangle |\lambda_j\rangle$$

Thus, now we apply  $U_{CR}$  to this state we get the following transformation:

$$|0\rangle \sum_{j=0}^{N-1} b_j |\tilde{\theta}_j\rangle |\lambda_j\rangle \xrightarrow{U_{CR} \otimes I} \sum_{j=0}^{N-1} b_j \left( \sqrt{1 - \frac{C^2}{\tilde{\theta}_j'^2}} |0\rangle + \frac{C}{\tilde{\theta}_j'} |1\rangle \right) |\tilde{\theta}_j\rangle |\lambda_j\rangle$$

where each  $\theta'_j \approx \lambda'_j$ .

4. Finally, we perform uncomputation by applying  $U_{QPE}^\dagger$ , we convert the information from the ancilla register  $|\tilde{\theta}_j\rangle$  back to  $|0\rangle_{n_l}$ . Thus, through the uncomputation, the  $n_l$  ancilla qubits for storing the eigenvalues also becomes a working register. Discarding all the working registers, the resulting unitary denoted by  $U_{HHL}$  satisfies

$$U_{HHL} |0\rangle |b\rangle = \sum_{j=0}^{N-1} \left( \sqrt{1 - \frac{C^2}{\tilde{\theta}_j'^2}} |0\rangle + \frac{C}{\tilde{\theta}_j'} |1\rangle \right) b_j |\lambda_j\rangle$$

Finally, measuring the signal qubit, if the outcome is 1, we obtain the (unnormalized) vector

$$\tilde{x} = \sum_j \frac{C b_j}{\tilde{\theta}_j'} |\lambda_j\rangle$$

stored as a normalized state in the system register is

$$|\tilde{x}\rangle = \frac{\tilde{x}}{\|\tilde{x}\|} \approx |x\rangle$$

which is the desired approximate solution to QLSP. In particular, the constant C does not appear in the solution.

**Recovering the norm of the solution:** The HHL algorithm returns the solution to QLSP in the form of a normalized state  $|x\rangle$  stored in the quantum computer. In order to recover the magnitude of the unnormalized solution  $\|\tilde{x}\|$ , we note that the success probability of measuring the signal qubit in 1 is

$$p(1) = \left\| \sum_j \frac{C b_j}{\tilde{\theta}_j'} \right\|^2 = \|\tilde{x}\|^2$$

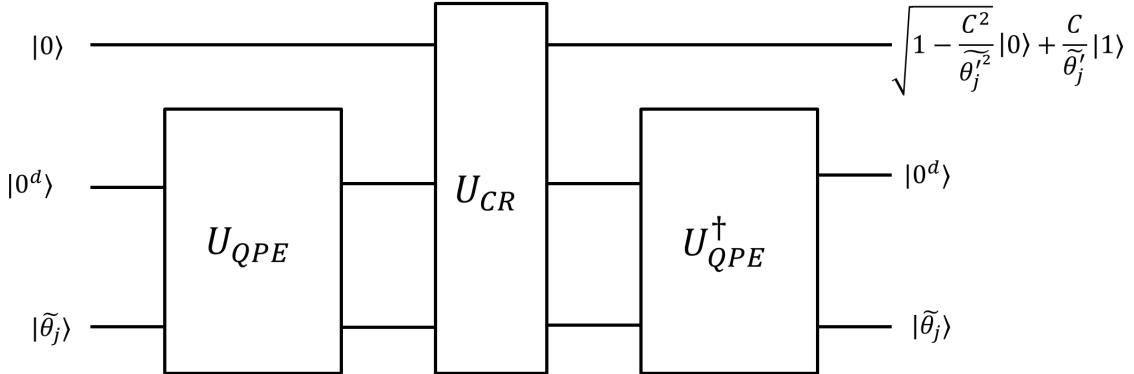


Figure 14.5: HHL Circuit

Therefore sufficient repetitions of running the circuit and estimate  $p(1)$ , we can obtain an estimate of  $\|\tilde{x}\|$ . The general HHL circuit is as shown in the figure 14.5.

### 14.3.3 Example

**Example 14.3.1.** (4-qubit HHL) Consider an example to illustrate the algorithm. Consider

$$A = \begin{bmatrix} 1 & -1/3 \\ -1/3 & 1 \end{bmatrix} \quad \text{and} \quad |b\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Note that  $b$  is normalized and  $b \in \mathbb{C}^2$ . Thus, we need  $n_b = 1$  qubit to represent  $b$ . Also,  $A \in \mathbb{C}^{2 \times 2}$  is Hermitian, and hence will have two real eigen values ( $\lambda_1, \lambda_2 \in \mathbb{R}$ ), thus we use  $n_l = 2$  qubits to store the binary representation of the eigen values and finally 1 auxiliary qubit for the conditioned rotation. Here,  $N = 2$ .

1. We first prepare the state  $|b\rangle$ . Note that here in this example  $|b\rangle = |0\rangle$ , trivial.

$$|0\rangle_{n_b} \xrightarrow{U_b} |b\rangle_{n_b}$$

2. Now we apply QPE, with inputs as  $|b\rangle_{n_b}$  and  $|0\rangle_{n_l}$ .

(Note that the eigen values of  $A$  are  $2/3$  and  $4/3$ . and suppose we choose  $t = 3/8$ , for simplicity. Thus, the eigen values of scaled  $A$  are:  $(2/3)t = (2/3)(3/8) = 2/8 = 1/4 = 0.25 = (0.01)$ ,  $(4/3)t = (4/3)(3/8) = 4/8 = 1/2 = 0/5 = (0.10)$ , thus the eigen values of scaled  $A$  in binary are  $0.01, 0.10$ . The

corresponding eigen vectors of A are

$$|\lambda_1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad \text{and} \quad |\lambda_2\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Thus, we can write  $|b\rangle$  in the eigen basis of A as

$$|b\rangle_{n_b} = \frac{1}{\sqrt{2}} \left( \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right) + \frac{1}{\sqrt{2}} \left( \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right) = \frac{1}{\sqrt{2}} |\lambda_1\rangle + \frac{1}{\sqrt{2}} |\lambda_2\rangle$$

$b_1 = \frac{1}{\sqrt{2}}, b_2 = \frac{1}{\sqrt{2}}$ . Note that all of this done here is not required since the algorithm will itself output the eigen values. This is done just for the sake of understanding and simplicity.)

Thus, QPE will output,

$$|0\rangle |0\rangle_{n_l} |b\rangle_{n_b} \xrightarrow{I \otimes QPE} |0\rangle \sum_{j=0}^1 b_j |\tilde{\theta}_j\rangle |\lambda_j\rangle = \frac{1}{\sqrt{2}} |0\rangle |01\rangle |\lambda_1\rangle + \frac{1}{\sqrt{2}} |0\rangle |10\rangle |\lambda_2\rangle$$

3. Now we do controlled rotations. (Suppose we use  $C = 1/8$  that is less than the smallest (rescaled) eigen values of A i.e.  $1/4$ , recall, C should be chosen such that it is less than the smallest (rescaled) eigenvalue of  $1/4$  but as large as possible so that when the auxiliary qubit is measured, the probability of being in the state  $|1\rangle$  is large.) Thus, as output of the controlled rotations we get,

$$\begin{aligned} & |0\rangle \sum_{j=0}^{N-1} b_j |\tilde{\theta}_j\rangle |\lambda_j\rangle \xrightarrow{U_{CR} \otimes I} \sum_{j=0}^{N-1} b_j \left( \sqrt{1 - \frac{C^2}{\tilde{\theta}_j'^2}} |0\rangle + \frac{C}{\tilde{\theta}_j'} |1\rangle \right) |\tilde{\theta}_j\rangle |\lambda_j\rangle \\ &= \frac{1}{\sqrt{2}} \left( \sqrt{1 - \frac{(1/8)^2}{(1/4)^2}} |0\rangle + \frac{1/8}{1/4} |1\rangle \right) |01\rangle |\lambda_1\rangle + \frac{1}{\sqrt{2}} \left( \sqrt{1 - \frac{(1/8)^2}{(1/2)^2}} |0\rangle + \frac{1/8}{1/2} |1\rangle \right) |10\rangle |\lambda_2\rangle \\ &= \frac{1}{\sqrt{2}} \left( \frac{\sqrt{3}}{2} |0\rangle + \frac{1}{2} |1\rangle \right) |01\rangle |\lambda_1\rangle + \frac{1}{\sqrt{2}} \left( \frac{\sqrt{15}}{4} |0\rangle + \frac{1}{4} |1\rangle \right) |10\rangle |\lambda_2\rangle \end{aligned}$$

4. Now we apply the uncomputation step  $QPE^\dagger$ , it will convert  $\tilde{\theta}_j$  to  $|0\rangle_{n_l}$ , (thus discarding the working registers as they will not contribute anything now to the algorithm) we will get the final state of the system to be in,

$$U_{HHL} |0\rangle |b\rangle = \sum_{j=0}^{N-1} \left( \sqrt{1 - \frac{C^2}{\tilde{\theta}_j'^2}} |0\rangle + \frac{C}{\tilde{\theta}_j'} |1\rangle \right) b_j |\lambda_j\rangle$$

Thus, the state of the system after the uncomputation will be:

$$\frac{1}{\sqrt{2}} \left( \frac{\sqrt{3}}{2} |0\rangle + \frac{1}{2} |1\rangle \right) |\lambda_1\rangle + \frac{1}{\sqrt{2}} \left( \frac{\sqrt{15}}{4} |0\rangle + \frac{1}{4} |1\rangle \right) |\lambda_2\rangle$$

Upon substituting,  $|\lambda_1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$

$|\lambda_2\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ , we get,

$$\begin{aligned} &= \frac{1}{\sqrt{2}} \left( \frac{\sqrt{3}}{2} |0\rangle + \frac{1}{2} |1\rangle \right) \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) + \frac{1}{\sqrt{2}} \left( \frac{\sqrt{15}}{4} |0\rangle + \frac{1}{4} |1\rangle \right) \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ &= \frac{\sqrt{3}}{4} |00\rangle - \frac{\sqrt{3}}{4} |01\rangle + \frac{1}{4} |10\rangle - \frac{1}{4} |11\rangle + \frac{\sqrt{15}}{8} |00\rangle + \frac{\sqrt{15}}{8} |01\rangle + \frac{1}{8} |10\rangle + \frac{1}{8} |11\rangle \end{aligned}$$

Thus, upon partial measurement of the auxiliary qubit (i.e. the first qubit here) to outcome 1, the state of the system will be:

$$\frac{\frac{1}{4} |10\rangle - \frac{1}{4} |11\rangle + \frac{1}{8} |10\rangle + \frac{1}{8} |11\rangle}{\sqrt{\frac{5}{32}}} = \frac{\frac{3}{8} |10\rangle - \frac{1}{8} |11\rangle}{\sqrt{\frac{5}{32}}} = |1\rangle \otimes \frac{\frac{3}{8} |0\rangle - \frac{1}{8} |1\rangle}{\sqrt{\frac{5}{32}}}$$

which is the value of normalized  $|x\rangle$  in the second register. Thus,

$$\frac{|x\rangle}{\|x\|\rangle} = \frac{\frac{3}{8} |0\rangle - \frac{1}{8} |1\rangle}{\sqrt{\frac{5}{32}}}$$

5. In order to calculate the norm of  $|x\rangle$ , we can perform several runs of the circuit and get the approximate  $p(1)$ . Here, the probability  $p(1)$  we get,

$$P(|1\rangle) = \left(\frac{3}{8}\right)^2 + \left(\frac{-1}{8}\right)^2 = \frac{5}{32} = \|x\|^2$$

In each of the run we can also measure the second register to get the approximate values of the components of  $x$  i.e.  $p(0) \approx \left(\frac{3/8}{\sqrt{5/32}}\right)^2 = \frac{9}{10}$  and  $p(1) \approx \left(\frac{-1/8}{\sqrt{5/32}}\right)^2 = \frac{1}{10}$ . Thus, the components will be the root of the probabilities and then multiplying this by the norm to recover the components of

the original vector  $x$ , we get,

$$\sqrt{\frac{5}{32}} \left( \sqrt{\frac{9}{10}} \right) = \frac{3}{8}; \quad \sqrt{\frac{5}{32}} \left( \sqrt{\frac{1}{10}} \right) = \frac{1}{8}$$

Now recall that we chose  $t = 3/8$  and  $C = 1/8$ , hence the current  $x$  is the solution of  $At$ . Thus, in order to recover the original  $x$ , we are required to multiply  $\frac{t}{C}$  to each of the components. Thus, we get,

$$\frac{3/8}{1/8} \left( \frac{3}{8} \right) = \frac{9}{8}; \quad \frac{3/8}{1/8} \left( \frac{1}{8} \right) = \frac{3}{8}$$

Thus, we get,

$$x = \begin{bmatrix} 9/8 \\ 1/8 \end{bmatrix}$$

Now, checking the value of  $x$  by substituting in  $Ax$ , we get,

$$\begin{bmatrix} 1 & -1/3 \\ -1/3 & 1 \end{bmatrix} \begin{bmatrix} 9/8 \\ 3/8 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |b\rangle$$

Thus, clearly, we have recovered the original solution  $x$ .

**Note that here we multiply the components by  $\frac{t}{C}$  can be explained as follows:**

**1. Multiplication by  $\frac{1}{C}$ , because recall that we are required to find**

$$|x\rangle = \sum_{j=0}^{N-1} \lambda_j^{-1} b_j |\lambda_j\rangle$$

**but using the algorithm we obtained**

$$|x\rangle = \sum_{j=0}^{N-1} \lambda_j^{-1} C b_j |\lambda_j\rangle$$

**Thus, upon multiplying the components by  $\frac{1}{C}$ , we recover the original components.**

2. Multiplying by  $t$ : Recall that we are not finding the solution of  $A$  but solution of scaled  $A$  i.e.  $At$ .

$$At|x'\rangle = |b\rangle$$

Thus, the original solution will be  $|x\rangle = t|x'\rangle$ . Thus, the multiplication by  $t$ .

Hence, upon multiplying the components of the square root of the approximate probabilities obtained from the algorithm by  $\frac{t}{C}$ , we recover the original components of  $x$ .

## 14.4 Complexity Analysis

Recall that, in the final step of the algorithm we get,

$$\tilde{x} = \sum_{j=0}^{N-1} \frac{Cb_j}{\tilde{\theta}'_j} |\lambda_j\rangle$$

Thus, in the register it will be stored as:

$$|\tilde{x}\rangle = \frac{\tilde{x}}{\|\tilde{x}\|} \approx |x\rangle$$

We get the probability of getting outcome 1 is  $p(1)$  as:

$$\begin{aligned} p(1) &= \|\tilde{x}\|^2 = \left\| \sum_{j=0}^{N-1} \frac{Cb_j}{\tilde{\theta}'_j} |\lambda_j\rangle \right\|^2 \\ &= C^2 \left\| \sum_{j=0}^{N-1} \frac{b_j}{\tilde{\theta}'_j} |\lambda_j\rangle \right\|^2 \\ &\approx C^2 \|A^{-1}|b\rangle\|^2 \end{aligned}$$

Therefore the success probability is determined by

1. The choice of the normalization constant  $C$
2. The norm of the true solution  $\|x\| = \|A^{-1}|b\rangle\|$

To maximize the success probability,  $C$  should be chosen to be as large as possible (without exceeding  $\lambda_0$ ). So assuming that the exact knowledge of  $\lambda_0$ , we can choose  $C = \lambda_0$ . For a Hermitian positive definite matrix  $A$ , (recall that 2-norm (induced norm is spectral norm) of any matrix  $A$  is given as  $\|A\|_2 = \sigma_{max} = \sqrt{\lambda_{max}(A^\dagger A)}$ , thus, for a Hermitian matrix ( $A^\dagger = A$ ;  $A = Q\Lambda Q^\dagger \implies \|A\|_2 = \sqrt{\lambda_{max}(A^2)} = \sqrt{\lambda_{max}(Q\Lambda Q^\dagger Q\Lambda Q^\dagger)} = \sqrt{\lambda_{max}(Q\Lambda^2 Q^\dagger)} = |\lambda_{max}|$  where  $\lambda_{max}$  is the eigen value with the largest magnitude (irrespective of sign)) which is also positive definite (has all the eigen values  $\geq 0$ )  $A$ ,  $\|A\| = \lambda_{N-1}$ , and  $\|A^{-1}\| = \lambda_0^{-1}$ . For simplicity, assume the largest eigenvalue of  $A$  is  $\lambda_{N-1} = 1$ . Then the condition number of  $A$  is

$$\kappa = \|A\| \|A^{-1}\| = \frac{\lambda_{N-1}}{\lambda_0} = C^{-1}$$

Also, using the definition of norms ( $\|A^{-1}|b\rangle\| \geq \|A^{-1}\| \| |b\rangle\|$ )

$$\|A^{-1}|b\rangle\| \geq \|A^{-1}\| \| |b\rangle\| = 1$$

Therefore,

$$\begin{aligned} p(1) &= C^2 \|A^{-1}|b\rangle\|^2 \geq \frac{1}{\kappa^2} \\ \implies p(1) &= \Omega(\kappa^{-2}) \end{aligned}$$

Thus, in the worst case we need to run the HHL algorithm for  $\mathcal{O}(\kappa^2)$  times to obtain the outcome 1 in the signal bit.

Assuming the number of qubits  $n$  is large, the circuit depth and the gate complexity of  $U_{HHL}$  is mainly determined by those of  $U_{QPE}$  because we assumed that it is easy to make  $|b\rangle$  and the only other part in the circuit is controlled rotation which has less number of gates than QFT in QPE. Thus, the majority of circuit depth and gate complexity is dominated by QPE of the HHL circuit. Therefore we can measure the complexity of the HHL algorithm in terms of the number of queries to  $U = e^{i2\pi A}$ .

In order to solve QLSP to precision  $\epsilon$ , we need to estimate the eigen values to multiplicative accuracy  $\epsilon$  instead of the standard additive accuracy. **An  $\epsilon$ -additive precision is an algorithm that for a true solution  $x$  returns  $\tilde{x} \in [x - \epsilon, x + \epsilon]$ . For a  $\epsilon$ -multiplicative precision algorithm, the algorithm returns  $\tilde{x} \in [x(1 + \epsilon), x(1 - \epsilon)]$ .** Assume  $\tilde{\theta}'_j = \lambda_j(1 + e_j)$  and  $|e_j| \leq \frac{\epsilon}{4} \leq \frac{1}{2}$ . Then the unnormalized solution satisfies

$$\|\tilde{x} - x\| = \left\| \sum_j b_j \left( \frac{1}{\tilde{\theta}'_j} - \frac{1}{\lambda_j} \right) |\lambda_j\rangle \right\| \leq \left\| \sum_j \frac{b_j}{\lambda_j} \left( \frac{-e_j}{1 + e_j} \right) |\lambda_j\rangle \right\| \leq \frac{\epsilon}{2} \|x\|$$

Hence, using the property that  $\|\vec{x}\| - \|\vec{y}\| \leq \|\vec{x} + \vec{y}\|$

$$\left|1 - \frac{\|\tilde{x}\|}{\|x\|}\right| \leq \frac{\epsilon}{2}$$

Then the normalized solution satisfies

$$\|\tilde{x}\rangle - |x\rangle\| = \left\|\frac{\tilde{x}}{\|\tilde{x}\|} - \frac{x}{\|x\|}\right\| \leq \left|1 - \frac{\|\tilde{x}\|}{\|x\|}\right| + \frac{\|\tilde{x} - x\|}{\|x\|} \leq \epsilon$$

The discussion requires the QPE to run to additive precision  $\epsilon' = \lambda_0\epsilon = \epsilon/\kappa$ . Therefore the query complexity of QPE is  $\mathcal{O}(\kappa/\epsilon)$ . Counting the number of times needed to repeat the HHL circuit, the worst case query complexity of the HHL algorithm is  $\mathcal{O}(\kappa^3/\epsilon)$ .

The above analysis is the worst case analysis, because we assume  $p(1)$  attains the lower bound  $\Omega(\kappa^{-2})$ . In practical applications, the result may not be so pessimistic. For instance, if  $b_j$  concentrates around the smallest eigen values of  $A$ , then we may have  $\|x\| \approx \Theta(\lambda_0^{-1} = \Theta(\kappa^{-1}))$ . Then  $p(1) = \Theta(1)$ . In such a case, we only need to repeat the HHL algorithm for a constant number of times to yield outcome 1 in the ancilla qubit. This does not reduce the query complexity of each run of the algorithm. Then in this best case, the query complexity is  $\mathcal{O}(\kappa/\epsilon)$ .

**Additional Considerations:** Below we discuss a few more aspects of the HHL algorithm. The first observation is that the asymptotic worst-case complexity of the HHL algorithm can be generally improved using amplitude estimation.

*Remark.* (HHL with amplitude amplification).

$$U_{HHL}|0\rangle|b\rangle = \sqrt{p(1)}|1\rangle|\psi_{good}\rangle + \sqrt{1-p(1)}|0\rangle|\psi_{bad}\rangle, \quad |\psi_{good}\rangle = |\tilde{x}\rangle$$

Since  $|\psi_{good}\rangle$  is marked by a signal qubit, we may construct a signal qubit to construct a reflection operator with respect to the signal qubit. This is simply given by

$$R_{good} = Z \otimes I_n$$

The reflection with respect to the initial vector is

$$R_{\psi_0} = U_{\psi_0}(2|01+n\rangle\langle 0^{1+n}| - I)U_{\psi_0}^\dagger$$

where  $U_{\psi_0} = U_{HHL}(I_1 \otimes U_b)$ . Let  $G = R_{\psi_0}R_{good}$  be the Grover iterate. Then amplitude amplification allows us to apply  $G$  for  $\Theta(p(1)^{-\frac{1}{2}})$  times to boost the success probability of obtaining  $\psi_{good}$  with constant success probability. Therefore in the worst case when  $p(1) = \Theta(\kappa^{-2})$ , the number of repetitions is reduced to  $\mathcal{O}(\kappa)$ , and

the total runtime is  $\mathcal{O}(\kappa^2/\epsilon)$ . This query complexity is the commonly referred query complexity for the HHL algorithm. Note that as usual, amplitude amplification increases the circuit depth. However, the tradeoff is that the circuit depth increases from  $\mathcal{O}(\kappa/\epsilon)$  to  $\mathcal{O}(\kappa^2/\epsilon)$ .

So far our analysis, especially that based on QPE relies on the assumption that all  $\lambda_j$  eigen values have exact d-bit representation. Such an assumption is unrealistic and causes theoretical and practical difficulties. Refer [?] for a more detailed explanation.

*Remark.* (Comparison with classical iterative linear system solvers). Let us now compare the cost of the HHL algorithm to that of classical iterative algorithms. If  $A$  is n-bit Hermitian positive definite with condition number  $\kappa$ , and is d-sparse (i.e. each row/column of  $A$  has at most d nonzero entries), then each matrix vector multiplication  $Ax$  costs  $\mathcal{O}(dN)$  floating point operations. The number of iterations for the steepest descent (SD) algorithm is  $\mathcal{O}(\kappa \log \epsilon^{-1})$ , and this number can be significantly reduced to  $\mathcal{O}(\sqrt{\kappa} \log \epsilon^{-1})$  by the renowned conjugate gradient (CG) method. Therefore the total cost (or wall clock time) of SD and CG is  $\mathcal{O}(dN\kappa \log \epsilon^{-1})$  and  $\mathcal{O}(dN\sqrt{\kappa} \log \epsilon^{-1})$ , respectively.

On the other hand, the query complexity of the HHL algorithm, even after using the Amplitude amplification algorithm, is still  $\mathcal{O}(\kappa^2/\epsilon)$ . Such a performance is terrible in terms of both *kappa* and  $\epsilon$ . Hence the power of the HHL algorithm, and other QLSP solvers is based on that each application of  $A$  (in this case, using Unitary  $U$ ) is much faster. In particular, if  $U$  can be implemented with  $\text{poly}(n)$  gate complexity (also can be measured by the wall clock time), then the total gate complexity of the HHL algorithm (with Amplitude amplification) is  $\mathcal{O}(\text{poly}(n)\kappa^2/\epsilon)$ . When  $n$  is large enough, we expect that  $\text{poly}(n) \ll N = 2^n$  and the HHL algorithm would eventually yield an advantage. Nonetheless, for a realistic problem, the assumption that  $U$  can be implemented with  $\text{poly}(n)$  cost should be taken with a grain of salt and carefully examined.

*Remark.* (Readout problem of QLSP). By solving the QLSP, the solution is stored as a quantum state in the quantum computer. Sometimes the QLSP is only a subproblem of a larger application, so it is sufficient to treat the HHL algorithm (or other QLSP solvers) as a "quantum subroutine", and leave  $|x\rangle$  in the quantum computer. However, in many applications (such as the solution of the Poisson's equation) the goal is to solve the linear system. Then the information in  $|x\rangle$  must be recovered to a measurable classical output.

The most common case is to compute the expectation of some observable  $\langle \mathcal{O} \rangle = \langle x | \mathcal{O} | x \rangle \approx \langle \tilde{x} | \mathcal{O} | \tilde{x} \rangle$ . Assuming  $\langle \mathcal{O} \rangle = \Theta(1)$ . Then to reach additive precision  $\epsilon$  of the observable, the number of samples needed is  $\mathcal{O}(\epsilon^{-2})$ . On the other hand,

in order to reach precision  $\epsilon$ , the solution vector  $|\tilde{x}\rangle$  must be solved to precision  $\epsilon$ . Assuming the worst case analysis for the HHL algorithm, the total query complexity needed is

$$\mathcal{O}(\kappa^2/\epsilon) \times \mathcal{O}(\epsilon^{-2}) = \mathcal{O}(\kappa^2/\epsilon^3)$$

*Remark.* (Query complexity lower bound). The cost of a quantum algorithm for solving a generic QLSP scales at least as  $\Omega(\kappa(A))$  where  $\kappa(A) = \|A\| \|A^{-1}\|$  is the condition number of  $A$ . The proof is based on converting the QLSP into a Hamiltonian simulation problem, and the lower bound with respect to  $\kappa$  is proved via the “no-fast-forwarding” theorem for simulating generic Hamiltonians [?]. Nonetheless, for specific classes of hamiltonians, it may still be possible to develop fast algorithm to overcome this lower bound.

## 14.5 Improvements in HHL algorithm

## 14.6 Implementation using Qiskit

## 14.7 Applications

### 14.7.1 Poisson’s problem

Poisson’s equation is a partial differential equation that describes the distribution of a potential field in a region. The equation is given as:

$$\nabla^2 u + f = 0$$

where  $u$  is the potential field, and  $f$  is the source term. The Poisson’s equation can be solved using the finite difference method. The finite difference method approximates the derivatives in the equation using finite difference schemes and discretizes the region into a grid, and the potential field is calculated at the grid points. The finite difference method converts the Poisson’s equation into a linear system of equations, which can be solved using the HHL algorithm. Consider a toy problem of solving the Poisson’s equation in one dimension with Dirichlet Boundary conditions as:

$$-u''(x) = f(x) \quad x \in [0, 1] \quad u(0) = u(1) = 0$$

Using the central finite difference formula for discretizing the Laplacian operator on a uniform grid.

$$-\frac{\partial^2 u}{\partial x^2} = f(x) \quad x \in [0, 1] \quad u(0) = u(1) = 0$$

Now recall that the central difference formula for discretizing second order derivative is

$$\frac{\partial^2 u}{\partial x^2} = \left( \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} \right)$$

Substituting in the equation we get the discretized form of the above Poisson's equation as:

$$-\left( \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} \right) = f(x_i)$$

Since we already know the solution at the boundary points from dirichlet conditions, thus, ignoring the values at the boundary points and writing equations at the N interior points we get,

$$A = \frac{1}{h^2} \begin{bmatrix} 2 & -1 & 0 & \dots & 0 & 0 & 0 \\ -1 & 2 & -1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -1 & 2 & -1 \\ 0 & 0 & 0 & \dots & 0 & -1 & 2 \end{bmatrix}$$

where  $h = \frac{1}{N+1}$ ,  $x_i \forall i \in \{1, \dots, N\}$ . Let  $u_i = u(x_i)$  and  $f_i = f(x_i)$ . We assume that vector  $b$  is already normalized so that  $|f\rangle = f$ .

**Proposition 14.7.1.** (*Diagonalization of tridiagonal matrices*) Consider the following Hermitian toeplitz tridiagonal matrix:

$$A = \begin{bmatrix} a & \bar{b} & 0 & \dots & 0 & 0 & 0 \\ b & a & \bar{b} & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & b & a & \bar{b} \\ 0 & 0 & 0 & \dots & 0 & b & a \end{bmatrix} \in \mathbb{C}^{N \times N}$$

can be analytically diagonalized as

$$Av_k = \lambda_k v_k, \quad k = 1, \dots, N.$$

where  $(v_k)_j = v_{j,k}$  denotes the  $j$ th component of  $v_k$  eigen vector,  $j = 1, \dots, N$ ,  $b = |b|e^{i\theta}$ , and

$$\lambda_k = a + 2|b| \cos \frac{k\pi}{N+1}; \quad v_{j,k} = \sin \frac{jk\pi}{N+1} e^{ij\theta}$$

*Proof.* Now  $v_{j,k} = \sin \frac{jk\pi}{N+1} e^{\iota j\theta}$ . Thus, note that  $v_{0,k} = v_{N+1,k} = 0$ . Then direct matrix vector multiplication above shows that for any  $j = 1, \dots, N$ ,

$$(Av_k)_j = bv_{k,j-1} + av_{k,j} + \bar{b}v_{k,j+1}$$

Upon substituting the value of  $v_{k,j}$  we get,

$$\begin{aligned} (Av_k)_j &= b \sin \frac{(j-1)k\pi}{N+1} e^{\iota(j-1)\theta} + a \sin \frac{jk\pi}{N+1} e^{\iota j\theta} + \bar{b} \sin \frac{(j+1)k\pi}{N+1} e^{\iota(j+1)\theta} \\ &= a \sin \frac{jk\pi}{N+1} e^{\iota j\theta} + |b| e^{\iota\theta} \sin \frac{(j-1)k\pi}{N+1} e^{\iota(j-1)\theta} + |b| e^{-\iota\theta} \sin \frac{(j+1)k\pi}{N+1} e^{\iota(j+1)\theta} \\ &= a \sin \frac{jk\pi}{N+1} e^{\iota j\theta} + |b| \left( \sin \frac{(j-1)k\pi}{N+1} + \sin \frac{(j+1)k\pi}{N+1} e^{\iota j\theta} \right) \\ &= a \sin \frac{jk\pi}{N+1} e^{\iota j\theta} + 2|b| \sin \frac{jk\pi}{N+1} \cos \frac{k\pi}{N+1} e^{\iota j\theta} \\ &= \left( a + 2|b| \cos \frac{k\pi}{N+1} \right) \sin \frac{jk\pi}{N+1} e^{\iota j\theta} \\ &= \lambda_k v_{j,k} \\ &= \lambda_k (v_k)_j \end{aligned}$$

□

Using this proposition and substituting with  $a = 2/h^2$ ,  $b = -1/h^2$ , the largest eigen value of  $A$  is  $\lambda_{max} = \|A\| \approx \frac{4}{h^2}$ , and the smallest eigen value  $\lambda_{min} = \|A^{-1}\|^{-1} \approx \pi^2$ . So,

$$\kappa(A) = \|A\| \|A^{-1}\| \approx \frac{4}{h^2 \pi^2} = \mathcal{O}(N^2)$$

Recall that the circuit depth of the HHL algorithm is  $\mathcal{O}(N^2/\epsilon)$ , and the worst case query complexity (using AA) is  $\mathcal{O}(N^4/\epsilon)$ . So when  $N$  is large, there is little benefit in employing the quantum computer to solve this problem.

Let us now consider solving a d-dimensional Poisson's equation with Dirichlet boundary conditions

$$-\nabla^2 u(r) = b(r), \quad r = \Omega = [0, 1]^d, \quad u|_{\partial\Omega} = 0$$

The grid is the Cartesian product of the uniform gird in 1D with  $N$  grid points per dimension and  $h = 1/(N+1)$ . The total number of grid points is  $N = N^d$ . After discretization we will obtain a linear system  $Au = b$ , where

$$A = A \otimes I \otimes I \dots \otimes I + I \otimes A \otimes I \dots \otimes I + \dots + I \otimes I \dots \otimes A$$

where  $I$  is the identity matrix of size  $N$ . Since  $A$  is Hermitian and positive definite, we have  $\|A\| \approx 4d/h^2$ , and  $\|A^{-1}\|^{-1} = d\pi^2$ . So  $\kappa(A) \approx 4/(h^2\pi^2) \approx \kappa(A)$ . Therefore the condition number is independent of the spatial dimension  $d$ .

The worst case query complexity of the HHL algorithm is  $\mathcal{O}(N^2/\epsilon)$ . So when the number of grid points per dimension  $N$  is fixed and the spatial dimension  $d$  increases, and if  $U = e^{\iota A\tau}$  can be implemented efficiently for some  $\tau\|A\| < 1$  with  $\text{poly}(d)$  cost, then the HHL algorithm will have an advantage over classical solvers, for which each matrix-vector multiplication scales linearly with  $N$  and is therefore exponential in  $d$ .

### 14.7.2 Solve Linear Differential Equations

Consider the solution of a time-dependent linear differential equation

$$\frac{d}{dt}x(t) = A(t)x(t) + b(t), \quad t \in [0, T]$$

$$x(0) = x_0$$

Here  $b(t), x(t) \in \mathbb{C}^d$  and  $A(t) \in \mathbb{C}^{d \times d}$ . The above differential equation can be used to represent a very large class of ordinary differential equations (ODEs) and spatially discretized partial differential equations (PDEs). For example, if  $A(t) = -\iota H(t)$  for some Hermitian Matrix  $H(t)$  and  $b(t) = 0$ , this is the time-dependent Schrodinger equation

$$\iota \frac{d}{dt}x(t) = H(t)x(t)$$

A special case when  $H(t) = H$  is often called the **Hamiltonian Simulation Problem**, and the solution can be written as

$$x(T) = e^{\iota HT}x(0)$$

which can be viewed as the problem of evaluating the matrix functions  $e^{\iota HT}$ .

In this section we consider the general case, and for simplicity discretize the equation in time using the forward Euler method with a uniform grid  $t_k = k\Delta t$  where  $\Delta t = T/N$ ,  $k = 0, \dots, N$ . Let  $A_k = A(t_k)$ ,  $b_k = b(t_k)$ . The resulting discretized system become

$$\begin{aligned} x_{k+1} - x_k &= \Delta t(A_k x_k + b_k), \quad k = 1, \dots, N \\ x_{k+1} &= (A_k \Delta t + I)x_k + b_k \Delta t \\ x_{k+1} - (I + A_k \Delta t)x_k &= b_k \Delta t \\ -(I + A_k \Delta t)x_k + x_{k+1} &= b_k \Delta t \end{aligned}$$

Note that for  $t = 0$ , since we know that  $x(0) = x_0$ , thus the above equation reduces to

$$x_1 = (I + A_0\Delta t)x_0 + \Delta t b_0$$

which can be rewritten as

$$\begin{bmatrix} I & 0 & 0 & \dots & 0 & 0 \\ -(I + \Delta t A_1) & I & 0 & \dots & 0 & 0 \\ 0 & -(I + \Delta t A_2) & I & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -(I + \Delta t A_{N-1}) & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} (I + \Delta t A_0)x_0 + b_0 \Delta t \\ b_1 \Delta t \\ b_2 \Delta t \\ \vdots \\ b_{N-1} \Delta t \end{bmatrix}$$

or more compactly as a linear system of equations

$$\mathcal{A}x = b$$

Here  $I$  is the identity matrix of size  $d$ , and  $x \in \mathbb{R}^{Nd}$  encodes the entire history of the states.

To solve the equation as a QLSP, the right hand side  $b$  needs to be a normalized vector. This means we need to properly normalize  $x_0, b_k$  so that

$$\|b\|^2 = \|(I + \Delta t A_0)x_0 + \Delta t b_0\|^2 + \sum_{k \in [N-1]} (\Delta t)^2 \|b_k\|^2 = 1$$

In the limit  $\Delta t \rightarrow 0$ , this can be written as

$$\|b\|^2 = \|x_0\|^2 + \int_0^T \|b(t)\|^2 dt = 1$$

which is not difficult to satisfy as long as  $x_0, b(t)$  can be prepared efficiently using unitary circuits and  $\|x_0\|, \|b(t)\| = \Theta(1)$ .

To solve this equation using the HHL algorithm we need to estimate the condition number of  $\mathcal{A}$ . Note that  $\mathcal{A}$  is a block-diagonal matrix and in particular is not Hermitian. So we need to use the dilation method and solve the corresponding Hermitian problem.

### Scalar case

In order to estimate the condition number, for simplicity we first assume  $d = 1$  (i.e., this is a scalar ODE problem), and  $A(t) = a \in \mathbb{C}$  is a constant. Then

$$\mathcal{A} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ -(1+a\Delta t) & 1 & 0 & \dots & 0 & 0 \\ 0 & -(1+a\Delta t) & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -(1+a\Delta t) & 1 \end{bmatrix} \in \mathbb{C}^{N \times N}$$

Recall that matrix  $\mathcal{A}$  should be invertible. Thus, let  $\zeta = 1 + a\Delta t$ . When  $\operatorname{Re}(a) \leq 0$ , the absolute stability condition of the forward Euler method requires  $|1 + a\Delta t| = |\zeta| < 1$ . In general we are interested in the regime  $\Delta t|a| \ll 1$ , and in particular  $|1 + a\Delta t| = |\zeta| < 2$ .

**Proposition 14.7.2.** *For any  $A \in \mathbb{C}^{N \times N}$ ,  $\|A\|^2, (1/\|A^{-1}\|)^2$  are given by the largest and smallest eigen values of  $A^\dagger A$  respectively.*

We first need to compute

$$\mathcal{A}^\dagger \mathcal{A} = \begin{bmatrix} 1 + |\zeta|^2 & -\bar{\zeta} & \dots & \dots & 0 & 0 \\ -\zeta & 1 + |\zeta|^2 & -\bar{\zeta} & \dots & 0 & 0 \\ 0 & -\zeta & 1 + |\zeta|^2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 + |\zeta|^2 & -\bar{\zeta} \\ 0 & 0 & 0 & \dots & -\zeta & 1 \end{bmatrix}$$

**Theorem 14.7.3.** *(Greshgorin Disk Theorem) Let  $A \in \mathbb{C}^{N \times N}$  with entries  $a_{ij}$ . For each  $i = 1, \dots, N$ , define*

$$R_i = \sum_{j \neq i} |a_{ij}|$$

*Let  $D(a_{ii}, R_i) \subseteq \mathbb{C}$  be a closed disc centered at  $a_{ii}$  with radius  $R_i$ , which is called a Greshgorin disk. Then every eigen value of  $A$  lies within at least one of the Greshgorin discs  $D(a_{ii}, R_i)$ .*

Since  $\mathcal{A}^\dagger \mathcal{A}$  is Hermitian, we can restrict the Greshgorin discs to the real line so that  $D(a_{ii}, R_i) \subseteq \mathbb{R}$ . Then Greshgorin discs of the matrix  $\mathcal{A}^\dagger \mathcal{A}$  satisfy the bound

$$D(a_{11}, R_1) \subseteq [1 + |\zeta|^2 - |\zeta|, 1 + |\zeta|^2 + |\zeta|]$$

$$D(a_{ii}, R_i) \subseteq [1 + |\zeta|^2 - 2|\zeta|, 1 + |\zeta|^2 + 2|\zeta|], \quad i = 2, \dots, N-1$$

$$D(a_{NN}, R_N) \subseteq [1 - |\zeta|, 1 + |\zeta|]$$

Now using the proposition we have, for  $D(a_{ii}, R_i) \subseteq [1 + |\zeta|^2 - 2|\zeta|, 1 + |\zeta|^2 + 2|\zeta|]$ ,  $i = 2, \dots, N-1$ . Thus, the maximum eigen value is

$$\lambda_{\max}(\mathcal{A}^\dagger \mathcal{A}) \leq (1 + |\zeta|)^2 < 9$$

for all values of  $a$  such that  $|\zeta| < 2$ .

To obtain a meaningful lower bound of  $\lambda_{\min}(\mathcal{A}^\dagger \mathcal{A})$ , we need  $\operatorname{Re}(a) < 0$  and hence  $|\zeta| < 1$ . Use the inequality

$$\begin{aligned} \sqrt{1+x} &\leq 1 + \frac{1}{2}x, \quad x > -1 \\ \implies -\sqrt{1+x} &\geq -1 - \frac{1}{2}x, \quad x > -1 \\ \implies 1 - \sqrt{1+x} &\geq -\frac{1}{2}x. \quad x > -1 \end{aligned}$$

we have

$$\begin{aligned} 1 - |\zeta| &= 1 - \sqrt{(1 + \Delta t \operatorname{Re}(a))^2 + (\Delta t)^2 (\operatorname{Im}(a))^2} \geq -\frac{1}{2}((\Delta t)^2 (\operatorname{Re}(a))^2 + 2\operatorname{Re}(a)\Delta t + (\Delta t)^2 (\operatorname{Im}(a))^2) \\ 1 - |\zeta| &\geq -\operatorname{Re}(a)\Delta t - \frac{1}{2}(\Delta t)^2 |a|^2 \geq -\frac{\Delta t}{2} \operatorname{Re}(a) \end{aligned}$$

when  $(\Delta t)^2 |a|^2 < (-\Delta t \operatorname{Re}(a))$  is satisfied. Then we have

$$1 > 1 - |\zeta| \geq -\frac{\Delta t}{2} \operatorname{Re}(a)$$

Now using the proposition we have, for  $D(a_{ii}, R_i) \subseteq [1 + |\zeta|^2 - 2|\zeta|, 1 + |\zeta|^2 + 2|\zeta|]$ ,  $i = 2, \dots, N-1$ . Thus, the minimum eigen value is

$$\lambda_{\min}(\mathcal{A}^\dagger \mathcal{A}) \geq (1 - |\zeta|)^2 \geq \frac{(\Delta t \operatorname{Re}(a))^2}{4}$$

Therefore

$$\|A\| = \sqrt{\lambda_{\max}(\mathcal{A}^\dagger \mathcal{A})} \leq \sqrt{1 + |\zeta|^2 + 2|\zeta|} < 3$$

and

$$\|A^{-1}\|^{-1} = \sqrt{\lambda_{\min}(\mathcal{A}^\dagger \mathcal{A})} \geq \frac{\Delta t(-\operatorname{Re}(a))}{2}$$

As a result, when  $(-Re(a)) = \Theta(1)$ , the condition number satisfies

$$\kappa(\mathcal{A}) = \|\mathcal{A}\| \|\mathcal{A}^{-1}\| = \mathcal{O}(1/\Delta t)$$

In summary, for the scalar problem  $d = 1$  and  $Re(a) < 0$ , the query complexity of the HHL algorithm is  $\mathcal{O}((\Delta t)^{-2}\epsilon^{-2})$ .

*Remark.* It may be tempting to modify the  $(N,N)$  th entry of  $\mathcal{A}$  to be  $1 + |\zeta|^2$  to obtain

$$\mathcal{G} = \begin{bmatrix} 1 + |\zeta|^2 & -\bar{\zeta} & 0 & \dots & 0 & 0 \\ -\zeta & 1 + |\zeta|^2 & -\bar{\zeta} & \dots & 0 & 0 \\ 0 & -\zeta & 1 + |\zeta|^2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 + |\zeta|^2 & -\bar{\zeta} \\ 0 & 0 & 0 & \dots & -\zeta & 1 + |\zeta|^2 \end{bmatrix}$$

Here  $\mathcal{G}$  is a Toeplitz tridiagonal matrix satisfying the requirements of Proposition 7.7.1. The eigen values of  $\mathcal{G}$  take the form

$$\lambda_k = 1 + |\zeta|^2 + 2|\zeta| \cos \frac{k\pi}{N+1}; \quad k = 1, \dots, N$$

If we take the approximation  $\lambda_{min}(\mathcal{A}^\dagger \mathcal{A}) \approx \lambda_{min}(\mathcal{G})$ , we would find that the above equation holds even when  $Re(a) > 0$ . This behavior is however incorrect, despite that the matrices  $\mathcal{A}^\dagger \mathcal{A}$  and  $\mathcal{G}$  only differ by a single entry!

### Vector Case

Here we consider a general  $d > 1$ , but for simplicity assume  $A(t) = A \in \mathbb{C}^{d \times d}$  is a constant matrix. We also assume  $A$  is diagonalizable with eigenvalue decomposition

$$A = V\Lambda V^{-1}$$

and  $\Lambda = diag(\lambda_1, \dots, \lambda_N)$ . We only consider the case  $Re(\lambda_k) < 0$  for all  $k$ .

**Proposition 14.7.4.** *For any diagonalizable  $A \in \mathbb{C}^{N \times N}$  with eigenvalue decomposition  $Av_k = \lambda v_k$ , we have*

$$\|A^{-1}\|^{-1} \leq \min_k |\lambda_k| \leq \max_k |\lambda_k| \leq \|A\|$$

*Proof.* Use the Schur form  $A = QTQ^\dagger$ , where  $Q$  is an unitary matrix and  $T$  is an upper triangular matrix. The diagonal entries of  $T$  encodes all eigenvalues of  $A$ , and the eigenvalues can appear in any order along the diagonal of  $T$ . The proposition follows by arranging the eigenvalue of  $A$  with the smallest and largest absolute values to the  $(N,N)$ th entry, respectively.  $\square$

The absolute stability condition of the forward Euler method requires  $\Delta t \|A\| < 1$ , and we are interested in the regime  $\Delta t \|A\| \ll 1$ . Therefore  $\Delta t |\lambda_k| \ll 1$  for all  $k$ . Let  $I$  be the identity matrix of size  $d$ , and denote  $B = -(I + \Delta t A)$ , then

$$\mathcal{A}^\dagger \mathcal{A} = \begin{bmatrix} I + B^\dagger B & B & \dots & 0 & 0 & 0 \\ B & I + B^\dagger B & B^\dagger & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & I + B^\dagger B & B^\dagger \\ 0 & 0 & 0 & \dots & B & I \end{bmatrix}$$

Note that

$$\|B\| \leq \|I + B^\dagger B\| \leq 1 + (1 + \Delta t \|A\|)^2 \leq 5$$

# Chapter 15

## Trotter based Hamiltonian Simulation

### 15.1 The Problem Definition

The Hamiltonian simulation problem with a time-independent Hamiltonian, or the Hamiltonian simulation problem for short is the following problem:

**Definition 15.1.1.** Given an initial state  $|\psi_0\rangle$  and a Hamiltonian  $H$ , evaluate the quantum state at time  $t$  according to  $|\phi(t)\rangle = e^{itH} |\psi_0\rangle$ .

Hamiltonian simulation is of immense importance in characterizing quantum dynamics for a diverse range of systems and situations in quantum physics, chemistry and materials science. Simulation of one quantum Hamiltonian by another quantum system was also one of the motivations of Feynman's 1982 proposal for design of quantum computers. We have also seen that Hamiltonian simulation appears as a quantum subroutine in numerous other quantum algorithms such as QPE and its various applications.

The Hamiltonian simulation problem can also be viewed as a linear ODE:

$$\partial_t \psi(t) = -\imath H \psi(t), \quad \psi(0) = \psi_0$$

However, thanks to the unitary operator  $e^{-\imath tH}$  for any  $t$ , we do not need to store the full history of the quantum states, and can instead focus on the quantum state at time  $t$  of interest.

Following conceptualization of a universal quantum simulator using a Trotter decomposition of time evolution operator  $e^{-\imath tH}$ , many new quantum algorithms for Hamiltonian simulation have been proposed. This chapter focuses on the Trotter based Hamiltonian simulation methods (also called the product formula).

## 15.2 Trotter Splitting

Consider the Hamiltonian simulation problem  $H = H_1 + H_2$ , where  $e^{-\imath H_1 \Delta t}$  and  $e^{-\imath H_2 \Delta t}$  can be efficiently computed at least for some  $\Delta t$ . In general  $[H_1, H_2] \neq 0$ , and the splitting of the evolution of  $H_1, H_2$  needs to be implemented via the Lie product formula

$$e^{-\imath t H} = \lim_{L \rightarrow \infty} \left( e^{-\imath \frac{t}{L} H_1} e^{-\imath \frac{t}{L} H_2} \right)^L$$

When taking  $L$  to be a finite number, and let  $\Delta t = t/L$ , this gives the simplest first order Trotter method with

$$\|e^{-\imath \Delta t H} - e^{-\imath \Delta t H_1} e^{-\imath \Delta t H_2}\| = \mathcal{O}(\Delta t^2)$$

Therefore to perform Hamiltonian simulation to time  $t$ , the error is

$$\|e^{-\imath t H} - \left( e^{-\imath \frac{t}{L} H_1} e^{-\imath \frac{t}{L} H_2} \right)^L \| = \mathcal{O}(\Delta t^2 L) = \mathcal{O}\left(\frac{t^2}{L}\right)$$

So to reach precision  $\epsilon$  in the operator norm, we need

$$L = \mathcal{O}(t^2 \epsilon^{-1})$$

This can be improved to the second order Trotter method (also called the symmetric trotter splitting or Strang splitting)

$$\|e^{-\imath \Delta t H} - e^{-\imath \Delta t / 2 H_2} e^{-\imath \Delta t H_1} e^{-\imath \Delta t / 2 H_2}\| = l = \mathcal{O}(\Delta t^3)$$

Following a similar analysis to the first order method, we find that to reach precision  $\epsilon$  we need

$$L = \mathcal{O}(t^{\frac{3}{2}} \epsilon^{-1/2})$$

Higher order Trotter methods are also available, such as the  $p$ -th order Suzuki formula. The local truncation error is  $(\Delta t)^{p+1}$ . therefore to reach precision  $\epsilon$ , we need

$$L = \mathcal{O}(t^{\frac{p+1}{p}} \epsilon^{-1/p})$$

This is often written as  $L = \mathcal{O}(t^{1+o(1)} \epsilon^{-o(1)})$  as  $p \rightarrow \infty$ .

**Example 15.2.1.** (Simulating transverse field Ising model). For the one dimensional transverse filed Ising model (TFIM) with nearest neighbor interaction, wince all Pauli- $Z_i$  operators commute, we have

$$e^{-\imath t H_1} := e^{\imath t} \sum_{i=1}^{n-1} Z_i Z_{i+1} = \prod_{i=1}^{n-1} e^{\imath t Z_i Z_{i+1}}$$

Each  $e^{itZ_i Z_{i+1}}$  is a rotation involving only the qubits  $i, j$ , and the splitting has no error. Similarly

$$e^{-itH_2} := e^{d \sum_i i X_i} = \prod_i e^{\imath t g X_i}$$

and each  $e^{\imath t g X_i}$  can be implemented independently without error.

**Example 15.2.2.** (Particle in a Potential). Let  $H = -\Delta_r + V(\mathbf{r}) = H_1 + H_2$  be the Hamiltonian of a particle in a potential field  $V(\mathbf{r})$ , where  $\mathbf{r} \in \Omega = [0, 1]^d$  with periodic boundary conditions. After discretization using Fourier modes,  $e^{\imath H_1 t}$  can be efficiently performed by diagonalizing  $H_1$  in the Fourier space, and  $e^{\imath H_2 t}$  can be efficiently performed since  $V(\mathbf{r})$  is diagonal in the real space.

## 15.3 Commutator type error bound

In this section we try to refine the error bounds in the equation by evaluating the preconstant explicitly. For simplicity we focus only on the first order Trotter formula. The trotter propagator  $\tilde{U}(t) = e^{-\imath t H_1} e^{-\imath t H_2}$  satisfies the equation

$$\begin{aligned} \imath \partial_t \tilde{U}(t) &= e^{-\imath t H_1} e^{-\imath t H_2} + e^{-\imath t H_1} H_2 e^{-\imath t H_2} \\ &= (H_1 + H_2) e^{-\imath t H_1} e^{-\imath t H_2} + e^{-\imath t H_1} H_2 e^{-\imath t H_2} - H_2 e^{-\imath t H_1} e^{-\imath t H_2} \\ &= H \tilde{U}(t) + [e^{-\imath t H_1}, H_2] e^{-\imath t H_2} \end{aligned}$$

with initial condition  $\tilde{U}(0) = I$ . By Duhamel's principle, and let  $U(t) = e^{-\imath t H}$ , we have

$$\tilde{U}(t) = U(t) - \imath \int_0^t e^{-\imath H(t-s)} [e^{-\imath s H_1}, H_2] e^{-\imath s H_2} ds$$

So we have

$$\|\tilde{U}(t) - U(t)\| \leq \int_0^t \|e^{-\imath s H_1}, H_2\| ds$$

Now consider  $G(t) = [e^{-\imath t H_1}, H_2] e^{\imath t H_1} = e^{-\imath t H_1} H_2 e^{\imath t H_1} - H_2$ , which satisfies  $G(0) = 0$  and

$$\imath \partial_t G(t) = e^{-\imath t H_1} [H_1, H_2] e^{\imath t H_1}$$

Hence,

$$\|[e^{-\imath t H_1}, H_2]\| = \|G(t)\| \leq t \| [H_1, H_2] \|$$

Plugging this back to Equation we have

$$\|\tilde{U}(9T) - U(t)\| \leq \int_0^t s \| [H_1, H_2] \| ds \leq \frac{t^2}{2} \| [H_1, H_2] \| \leq t^2 \nu^2$$

In the last equality, we have used the relation  $\|[H_1, H_2]\| \leq 2\nu^2$  with  $\nu = \max\{\|H_1\|, \|H_2\|\}$ . Therefore equation can be replaced by a sharper inequality

$$\|e^{-\iota\Delta t H} - e^{-\iota\Delta t H_1} e^{-\iota\Delta t H_2}\| \leq \frac{\Delta t^2}{2} \|[H_1, H_2]\| \leq (\Delta t)^2 \nu^2$$

Here the first inequality is called the commutator norm error estimate, and the second inequality the operator norm error estimate.

For the transverse field Ising model with nearest neighbor interaction, we have  $\|H_1\|, \|H_2\| = \mathcal{O}(n)$ , and hence  $\nu^2 = \mathcal{O}(n^2)$ . On the other hand, since  $[Z_i Z_j, X_k] \neq 0$  only if  $k = i$  or  $k = j$ , the commutator bound satisfies  $\|[H_1, H_2]\| = \mathcal{O}(n)$ . therefore to reach precision  $\epsilon$ , the scaling of the total number of time steps  $L$  with respect to the system size is  $\mathcal{O}(n^2/\epsilon)$  according to the estimate based on the operator norm, but is only  $\mathcal{O}(n/\epsilon)$  according to that based on the commutator norm.

For the particle in a potential, for simplicity consider  $d = 1$  and the domain  $\Omega = [0, 1]$  is discretized using a uniform grid of size  $N$ . For smooth and bounded potential, we have  $\|H_1\| = \mathcal{O}(N^2)$ , and  $\|V\| = \mathcal{O}(1)$ . therefore the operator norm bound gives  $\nu^2 = \mathcal{O}(N^4)$ . This is too pessimistic. Reexamining the second inequality of the equation shows that in this case, the error bound should be  $\mathcal{O}((\Delta t)^2 \nu)$  instead of  $(\Delta t)^2 \nu^2$ . So according to the operator norm error estimate, we have  $L = \mathcal{O}(N^2/\epsilon)$ . On the other hand, in the continuous space, for any smooth function  $\psi(r)$ , we have

$$[H_1, H_2]\psi = \left[ -\frac{d^2}{dr^2}, V \right] \psi = -V''\psi - V'\psi'$$

So

$$\|H_1, H_2\|\psi \leq V''\|\psi\| + \|V'\|\|\psi'\| = \mathcal{O}(N)$$

here we have used that  $\|V'\| = \|V''\| = \mathcal{O}(1)$ , and  $\|\psi'\| = \mathcal{O}(N)$  in the worst case scenario. therefore  $\|[H_1, H_2]\| = \mathcal{O}(N)$ , and we obtain a significantly improved estimate  $L = \mathcal{O}(N/\epsilon)$  according to the commutator norm.

The commutator scaling of the trotter error is an important feature of the method. there are papers for analysis of the second order Trotter method, and for the analysis of the commutator scaling of high order trotter methods.

*Remark.* (vector norm bound). The Hamiltonian simulation problem of interest in practice often concerns the solution with particular types of initial conditions, instead of arbitrary initial conditions. Therefore the operator norm bound in the equation can still be too loose. Taking the initial condition into account, we readily obtain

$$\|e^{-\iota|\Delta t| H} \psi(0) - e^{i\omega \Delta t H_1} e^{-\iota\Delta t H_2} \psi(0)\| \leq \frac{\Delta t^2}{2} \max_{0 \leq s \leq \Delta t} \|[H_1, H_2]\psi(s)\|$$

For the example of the particle in a potential, we have

$$\max_{0 \leq s \leq \Delta t} \| [H_1, H_2] \psi(s) \| \leq \|V''\| + \|V'\| \max_{0 \leq s \leq \Delta t} \|\psi'(s)\|$$

Therefore, if we are given the a priori knowledge that  $\max_{0 \leq s \leq t} \|\psi'(s)\| = \mathcal{O}(1)$ , we may even have  $L = \mathcal{O}(\epsilon^{-1})$ , i.e., the number of time steps is independent of  $N$ .

**Example 15.3.1.** Consider the Hamiltonian simulation problem for  $H = H_1 + H_2 + H_3$ . Show that the first order Trotter formula

$$\tilde{U}(t) = e^{-itH_1} e^{-itH_2} e^{-itH_3}$$

has a commutator type error bound.

**Example 15.3.2.** Consider the time-independent Hamiltonian simulation problem for the following controlled Hamiltonian

$$H(t) = a(t)H_1 + b(t)H_2$$

where  $a(t)$  and  $b(t)$  are smooth functions bounded together with all derivatives. We focus on the following trotter type splitting, defined as

$$\tilde{U}(t) := \tilde{U}(t_n, t_{n-1}) \dots \tilde{U}(t-1, t_0), \quad \tilde{U}(t_j+1, t_j) = e^{-\iota \Delta t a(t_j) H_1} e^{-\iota \Delta t b(t_j) H_2}$$

where the intervals  $[t_j, t_{j+1}]$  are equidistant and of length  $\Delta t$  on the interval  $[0, t]$  with  $t_n = t$ . Show that this method has first-order accuracy, but does not exhibit a commutator type error bound in general.

# Chapter 16

## Block Encoding

In order to perform matrix computations, we must first address the problem of the input model: how to get access to information in matrix  $A \in \mathbb{C}^{N \times N}$  ( $N = 2^n$ ) which is generally a non-unitary matrix, into the quantum computer? One possible input model is given via the unitary  $e^{i\tau A}$  (if  $A$  is not Hermitian, in some scenarios we can consider its Hermitian version via the dilation method).

A more general input model, as will be discussed in this chapter is called "block encoding". Of course, if  $A$  is a dense matrix without obvious structures, any input model will be very expensive (e.g. exponential in  $n$ ) to implement. Therefore a commonly assumed input model is  $s$ -sparse there are at most  $s$  nonzero entries in each row/column of the matrix. Furthermore, we have an efficient procedure to get access to the location, as well as the value of the nonzero entries. This in general can again be a difficult task given that the number of nonzero entries can still be exponential in  $n$  for a sparse matrix. Some dense matrices may also be efficiently block encoded on quantum computers. This chapter will illustrate the block encoding procedure via a number of detailed examples.

### 16.1 Query model for matrix entries

The query model for sparse matrices is based on certain quantum oracles. In some scenarios, these quantum oracles can be implemented all the way to the elementary gate level. Throughout the discussion we assume  $A$  is an  $n$ -qubit, square matrix, and

$$\|A\|_{max} = \max_{ij} |A_{ij}| < 1$$

If the  $\|A\|_{max} \geq 1$ , we can simply consider the rescaled matrix  $\tilde{A}/\alpha$  for some  $\alpha > \|A\|_{max}$ . To query the entries of a matrix, the desired oracle takes the following

general form

$$O_A |0\rangle |i\rangle |j\rangle = \left( A_{ij} |0\rangle + \sqrt{1 - |A_{ij}|^2} |1\rangle \right) |i\rangle |j\rangle$$

In other words, given  $i, j \in [N]$  and a signal bit 0,  $O_A$  performs a controlled rotation (controlling on  $|i\rangle |j\rangle$ ) of the signal bit, which encodes the information in terms of amplitude of  $|0\rangle$ .

However, the classical information in  $A$  is usually not stored natively in terms of such an oracle  $O_A$ . Sometimes it is more natural to assume that there is an oracle

$$\tilde{O}_A |0^{d'}\rangle |i\rangle |j\rangle = |\tilde{A}_{ij}\rangle |i\rangle |j\rangle$$

where  $\tilde{A}_{ij}$  is a  $d'$ -bit fixed point representation of  $A_{ij}$ , and the value of  $\tilde{A}_{ij}$  is either computed on the fly with a quantum computer, or obtained through an external database. In either case, the implementation of  $\tilde{O}_A$  may be challenging, and we will only consider the query complexity with respect to this oracle.

Using classical arithmetic operations, we can convert this oracle into an oracle

$$O'_A |0^d\rangle |i\rangle |j\rangle = |\tilde{\theta}_{ij}\rangle |i\rangle |j\rangle$$

where  $0 \leq \tilde{\theta}_{ij} < 1$ , and  $\tilde{\theta}_{ij}$  is a  $d$ -bit representation of  $\theta_{ij} = \arccos(A_{ij})/\pi$ . This step may require some additional work registers not shown here.

Now using the controlled rotation, the information of  $\tilde{A}_{ij}$ ,  $\tilde{\theta}_{ij}$  has now been transferred to the phase of the signal bit. We should then perform uncomputation and free the work register storing such intermediate information  $\tilde{A}_{ij}$ ,  $\tilde{\theta}_{ij}$ . The procedure is as follows:

$$\begin{aligned} |0\rangle |0^{d'}\rangle |i\rangle |j\rangle &\xrightarrow{O'_A} |0\rangle |\tilde{\theta}_{ij}\rangle |i\rangle |j\rangle \\ &\xrightarrow{CR} \left( A_{ij} |0\rangle + \sqrt{1 - |A_{ij}|^2} |1\rangle \right) |\tilde{\theta}_{ij}\rangle |i\rangle |j\rangle \\ &\xrightarrow{(O'_A)^{-1}} \left( A_{ij} |0\rangle + \sqrt{1 - |A_{ij}|^2} x |1\rangle \right) \text{ket}0^d |i\rangle |j\rangle \end{aligned}$$

From now on, we will always assume that the matrix entries of  $A$  can be queried using the phase oracle  $O_A$  or its variants.

## 16.2 Block Encoding

The simplest example of block encoding is the following: assume we can find a  $(n+1)$ qubit unitary  $U$  (i.e.  $U \in \mathbb{C}^{2N \times 2N}$ ) such that

$$U_A = \begin{bmatrix} A & * \\ * & * \end{bmatrix}$$

here \* means that the corresponding entries are irrelevant, then for any  $n$ -qubit quantum state  $|b\rangle$ , we can consider the state

$$|0, b\rangle = |0\rangle |b\rangle = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

and

$$U_A |0, b\rangle = \begin{bmatrix} Ab \\ * \end{bmatrix} = |0\rangle A |b\rangle + |\perp\rangle$$

Here the unnormalized state  $|\perp\rangle$  can be written as  $|1\rangle |\psi\rangle$  for some unnormalized state  $|\psi\rangle$ , that is irrelevant to the computation of  $A |b\rangle$ . In particular, it satisfies the orthogonality relation.

$$(\langle 0 | \otimes I_n) |\perp\rangle = 0$$

in order to obtain  $A |b\rangle$ , we need to ensure measure the qubit 0 and only keep the state if it returns 0. This can be summarised into the following quantum circuit as shown in the figure 16.1.

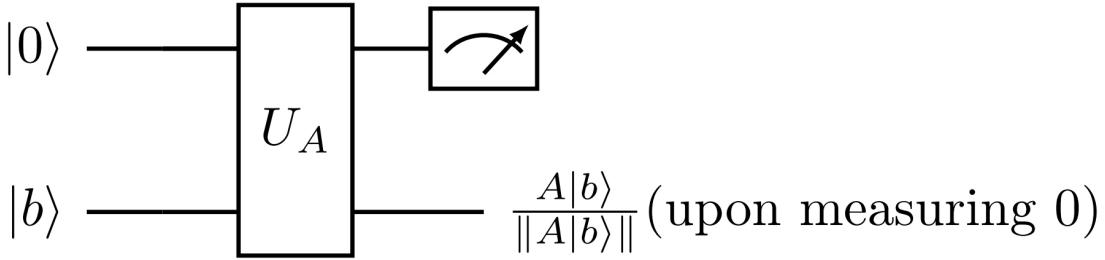


Figure 16.1: Circuit for Block Encoding A using one Ancilla qubit

Note that the output state is normalized after the measurement takes place. The success probability of obtaining 0 from the measurement can be computed as

$$p(0) = \|A |b\rangle\|^2 = \langle b | A^\dagger A |b\rangle$$

So the missing information of norm  $\|A|b\rangle\|$  can be recovered via the success probability  $p(0)$  if needed. We find that the success probability is only determined by  $A|b\rangle$ , and is independent of other irrelevant components of  $U_A$ .

Note that we may not need to restrict the matrix  $U_A$  to be a  $(n+1)$ -qubit matrix. If we can find any  $(n+m)$ -qubit matrix  $U_A$  so that

$$U_A = \begin{bmatrix} A & * & \dots & * \\ * & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \dots & * \end{bmatrix}$$

Here each  $*$  stands for an  $n$ -qubit matrix, and there are  $2^m$  block rows/columns in  $U_A$ . The relation above can be written compactly using the braket notation as

$$A = (\langle 0^m | \otimes I_n) U_A (| 0^m \rangle \otimes I_n)$$

since  $\langle 0^m | \otimes I_n$  will be a block row vector with Identity matrix (of size  $2^n \times 2^n$ ) at the first entry and 0 everywhere else. Similarly,  $| 0^m \rangle \otimes I_n$  will be a block column vector with Identity matrix at the first entry and 0 everywhere else. Thus, the above expression can be thought of as being simplified to

$$\langle 0^m | \otimes I_n = [1 \ 0 \ \dots \ 0] \otimes \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & 1 \end{bmatrix} = [I_n \ 0 \ 0 \ \dots \ 0]$$

where  $I_n$  is the  $n$ -qubit identity matrix of size  $2^n \times 2^n$  and 0 indicates block matrices of size  $2^n \times 2^n$  with all entries 0. Thus,  $\langle 0^m | \otimes I_n$  is a matrix of size  $2^n \times 2^{n+m}$ . Similarly for  $| 0^m \rangle \otimes I_n$ , we get,

$$| 0^m \rangle \otimes I_n = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & 1 \end{bmatrix} = \begin{bmatrix} I_n \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Thus, on putting it all together we get,

$$(\langle 0^m | \otimes I_n) U_A (|0^m\rangle \otimes I_n) = [I_n \ 0 \ 0 \ \dots \ 0] U_A \begin{bmatrix} I_n \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = A$$

A necessary condition for the existence of  $U_A$  is  $\|A\| \leq 1$ . (Note:  $\|A\|_{max} \leq 1$  does not guarantee that  $\|A\| \leq 1$ ). However, if we can find sufficiently large  $\alpha$  and  $U_A$  so that

$$A/\alpha = (\langle 0^m | \otimes I_n) U_A (|0^m\rangle \otimes I_n)$$

Measuring the  $m$  ancilla qubits and given that all  $m$ -qubits return 0, we still obtain the normalized state  $\frac{A|b\rangle}{\|A|b\rangle\|}$ . The number  $\alpha$  is hidden in the success probability:

$$p(0^m) = \frac{1}{\alpha^2} \|A|b\rangle\|^2 = \frac{1}{\alpha^2} \langle b|A^\dagger A|b\rangle$$

So if  $\alpha$  is chosen to be too large, the probability of obtaining all 0s from the measurement can be vanishingly small.

Finally, it can be difficult to find  $U_A$  to encode  $A$  exactly. This is not a problem, since it is sufficient if we can find  $U_A$  to block encode  $A$  upto some error  $\epsilon$ . We are now ready to give the definition of block encoding.

**Definition 16.2.1.** (Block encoding) Given an  $n$ -qubit matrix  $A$ , if we can find  $\alpha, \epsilon \in \mathbb{R}_+$ , and an  $(m+n)$ -qubit unitary matrix  $U_A$  so that

$$\|A - \alpha(\langle 0^m | \otimes I_n) U_A (|0^m\rangle \otimes I_n)\| \leq \epsilon$$

then  $U_A$  is called an  $(\alpha, m, \epsilon)$ -block encoding of  $A$ . When the block encoding is exact with  $\epsilon = 0$ ,  $U_A$  is called an  $(\alpha, m)$ -block encoding of  $A$ . The set of all  $(\alpha, m, \epsilon)$ -block-encoding of  $A$  is denoted by  $BE_{\alpha, m}(A, \epsilon)$ , and we define  $BE_{\alpha, m}A = BE(A, 0)$ .

Assume we know each matrix element of the  $n$ -qubit matrix  $A_{ij}$ , and we are given an  $(n+m)$ -qubit unitary  $U_A$ . In order to verify that  $U_A \in BE_{1,m}(A)$  we only need to verify that

$$\langle 0^m | \langle i | U_A | 0^m \rangle | j \rangle = A_{ij}$$

Here,  $|0^m\rangle |j\rangle$  will be a column vector of size  $(2^m \times 1 \otimes 2^n \times 1 = 2^{m+n} \times 1)$  which will be 1 in the  $j$ th row and 0 everywhere else. Similar arguments can be used for

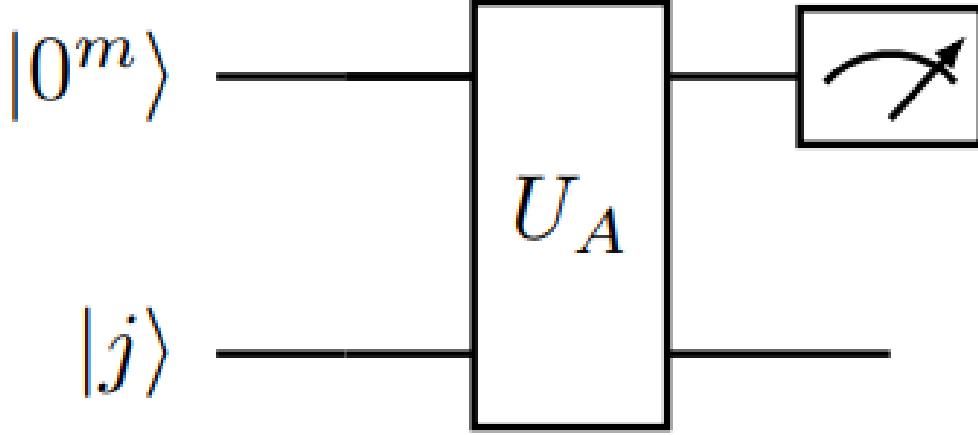


Figure 16.2: Circuit for general block encoding of A

establishing  $\langle 0^m | i \rangle$  which will be a row vector with 1 in the  $i$ th column entry and 0 everywhere else. Thus, upon multiplying the matrices it will pick the  $i, j$  th entry of the matrix  $U_A$  which will be  $A_{ij}$ .  $U_A$  applied to the vector  $|0^m, b\rangle$  can be obtained via the superposition principle.

Therefore we may first evaluate the state  $U_A |0^m, j\rangle$ , and perform inner product with  $|0^m, i\rangle$  and verify the resulting the inner product is  $A_{ij}$ . We will also use the following technique frequently. Assume  $U_A = U_B U_C$ , and then

$$\langle 0^m, i | U_A | 0^m, j \rangle = \langle 0^m, i | U_B U_C | 0^m, j \rangle = (U_B^\dagger | 0^m, i \rangle)^\dagger (U_C | 0^m, j \rangle)$$

So we can evaluate the states  $U_B^\dagger | 0^m, i \rangle$ ,  $U_C | 0^m, j \rangle$  independently, and then verify the inner product is  $A_{ij}$ . Such a calculation amounts to running the circuit shown in fig 16.2, and if the ancilla qubits are measured to be  $0^m$ , the system qubits return the normalized state  $\sum_i A_{ij} |i\rangle / \| \sum_i A_{ij} |i\rangle \|$ .

**Example 16.2.1.** ((1,1)-block-encoding in general). For any n-qubit matrix A with  $\|A\|_2 \leq 1$ , the singular value decomposition (SVD) of A is denoted by  $W\Sigma V^\dagger$ , where all singular values in the diagonal matrix  $\Sigma$  belong to  $[0, 1]$ . Then we may construct

an  $(n+1)$ -qubit unitary matrix

$$\begin{aligned} U_A &= \begin{bmatrix} W & 0 \\ 0 & I_n \end{bmatrix} \begin{bmatrix} \Sigma & \sqrt{I_n - \Sigma^2} \\ \sqrt{I_n - \Sigma^2} & -\Sigma \end{bmatrix} \begin{bmatrix} V^\dagger & 0 \\ 0 & I_n \end{bmatrix} \\ &= \begin{bmatrix} A & W\sqrt{I_n - \Sigma^2} \\ \sqrt{I_n - \Sigma^2}V^\dagger & -\Sigma \end{bmatrix} \end{aligned}$$

which is a  $(1, 1)$ -block-encoding of  $A$ .

**Example 16.2.2.** (Random circuit block encoded matrix). In some scenarios, we may want to construct a psuedo-random non-unitary matrix on quantum computers. Note that it would be highly inefficient if we first generate a dense psuedo-random matrix  $A$  classically and then feed it into the quantum computer using e.g. quantum-random-access memory (QRAM). Instead we would like to work with matrices that are inherently easy to generate on quantum computers. This inspires the random circuit based block encoding matrix (RACBEM) model. Instead of first identifying  $A$  and then finding its block encoding  $U_A$ , we reverse this thought process: we first identify a unitary  $U_A$  that is easy to implement on a quantum computer, and then ask which matrix can be block encoded by  $U_A$ .

This example shows that in principle, any matrix  $A$  with  $\|A\|_2 \leq 1$  can be accessed via a  $(1, 1, 0)$ -block-encoding. In other words,  $A$  can be block encoded by an  $(n+1)$ -qubit random unitary  $U_A$ , and  $U_A$  can be constructed using only one-qubit unitaries and CNOT gates. The layout of the two-qubit operations can be designed to be compatible with the coupling map of the hardware.

**Example 16.2.3.** (Block encoding of a diagonal matrix). As a special case, let us consider the block encoding of a diagonal matrix. Since the row and column indices are the same, we may simplify the oracle into

$$O_A |0\rangle |i\rangle = \left( A_{ii} |0\rangle + \sqrt{1 - |A_{ii}|^2} |1\rangle \right) |i\rangle$$

In the case when the oracle  $\tilde{O}_A$  is used, we may assume accordingly

$$\tilde{O}_A |0^{d'}\rangle |i\rangle = |\tilde{A}_{ii}\rangle |i\rangle$$

Let  $U_A = O_A$ . Direct calculation shows that for any  $i, j \in [N]$ ,

$$\langle 0 | \langle i | U_A | 0 \rangle | j \rangle = A_{ii} \delta_{ij}$$

This proves that  $U_A \in BE_{1,1}(A)$ i.e.,  $U_A$  is a  $(1, 1)$ -block-encoding of the diagonal matrix  $A$ .

## 16.3 Block Encoding of s-sparse matrix

We now give a few examples of block encodings of more general sparse matrices. We start from a 1-sparse matrix, i.e., there is only one nonzero entry in each row or column of the matrix. This means that for each  $j \in [N]$ , there is a unique  $c(j) \in [N]$  such that  $A_{c(j),j} \neq 0$ , and the mapping  $c$  is a permutation. Then there exists a unitary  $O_c$  such that

$$O_c |j\rangle = |c(j)\rangle$$

The implementation of  $O_c$  may require the usage of some work registers that are omitted here. We also have

$$O_c^\dagger |c(j)\rangle = |j\rangle$$

We assume the matrix entry  $A_{c(j),j}$  can be queried via

$$O_A |0\rangle |j\rangle = \left( A_{c(j),j} |0\rangle + \sqrt{1 - |A_{c(j),j}|^2} |1\rangle \right) |j\rangle$$

Now we construct  $U_A = (I \otimes O_c)O_A$ , (tensor product of two unitary matrices is unitary) and compute

$$\langle i| \langle 0| U_A |0\rangle |j\rangle = \langle 0| \langle i| \left( A_{c(j),j} |0\rangle + \sqrt{1 - |A_{c(j),j}|^2} |1\rangle \right) |c(j)\rangle = A_{c(j),j} \delta_{i,c(j)}$$

This proves that  $U_A \in BE_{1,1}(A)$ .

For a more general s-sparse matrix, WLOG we assume each row and column has exactly  $s$  nonzero entries (otherwise we can always treat some zero entries as nonzeros). For each column  $j$ , the row index for the  $l$ th nonzero entry is denoted by  $c(j, l) = c_{j,l}$ . For simplicity, we assume that there exists a unitary  $O_c$  such that

$$O_c |l\rangle |j\rangle = |l\rangle |c(j, l)\rangle$$

Here we assume  $s = 2^p$  and the first register is an  $p$ -qubit register. A necessary condition for this model is that  $O_c$  is reversible, i.e., we can have  $O_c^\dagger |l\rangle |c(j, l)\rangle = |l\rangle |j\rangle$ . This means that for each row index  $i = c(j, l)$ , we can recover the column index  $j$  given the value of  $l$ . This can be satisfied e.g.

$$c(j, l) = j + l - l_0 \pmod{N}$$

where  $l_0$  is a fixed number. This corresponds to a banded matrix. This assumption is of course somewhat restrictive. We shall discuss more general query models.

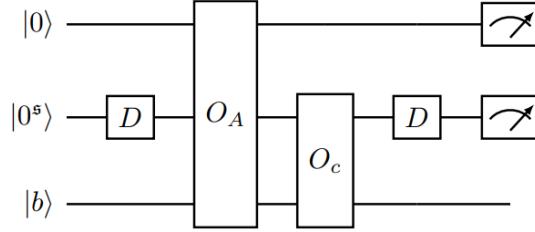


Figure 16.3: Quantum circuit for block encoding an s-sparse matrix

Corresponding to the equation, the matrix entries can be queried via

$$O_A |0\rangle |l\rangle |j\rangle = \left( A_{c(j,l),j} |0\rangle + \sqrt{1 - |A_{c(j,l),j}|^2} |1\rangle \right) |l\rangle |j\rangle$$

In order to construct a unitary that encodes all row indices at the same time ,we define  $D = H^{\otimes p}$  (sometimes called a diffusion operator, which is a term originated from Grover's search) satisfying

$$D |0^p\rangle = \frac{1}{\sqrt{p}} \sum_{l \in [p]} |l\rangle$$

Consider  $U_A$  given by the circuit in figure 16.3. The measurement means that to obtain  $A |b\rangle$ , the ancilla register should return the value 0.

**Proposition 16.3.1.** *The circuit given in figure 16.3 defines  $U_A \in BE_{s,s+1}(A)$ .*

*Proof.* We call  $|0\rangle |0^p\rangle |j\rangle$  the source state, and  $|0\rangle |0^p\rangle |i\rangle$  the target state. In order to compute the inner product  $\langle 0| \langle 0^s | i | U_A | 0 \rangle | 0^p \rangle | i \rangle$ , we apply  $D, O_A, O_C$  to the source state accordingly as

$$\begin{aligned} |0\rangle |0^p\rangle |j\rangle &\xrightarrow{D} \frac{1}{\sqrt{s}} \sum_{l \in [s]} |0\rangle |l\rangle |j\rangle \\ &\xrightarrow{O_A} \frac{1}{\sqrt{s}} \sum_{l \in [s]} \left( A_{c(j,l),j} |0\rangle + \sqrt{1 - |A_{c(j,l),j}|^2} |1\rangle \right) |l\rangle |j\rangle \\ &\xrightarrow{O_C} \frac{1}{\sqrt{s} \sum_{l \in [s]}} \left( A_{c(j,l),j} |0\rangle + \sqrt{1 - |A_{c(j,l),j}|^2} |1\rangle \right) |l\rangle |c(j,l)\rangle \end{aligned}$$

Since we are only interested in the final state when all ancilla qubits are the 0 state, we may apply  $D$  to the target state  $|0\rangle |0^p\rangle |i\rangle$  as (note that D is Hermitian)

$$|0\rangle |0^p\rangle |i\rangle \xrightarrow{D} \frac{1}{\sqrt{s}} \sum_{l' \in [s]} |0\rangle |l'\rangle |i\rangle$$

Hence the inner product

$$\langle 0 | \langle 0^p | \langle i | U_A | 0 \rangle | 0^p \rangle | j \rangle = \frac{1}{s} \sum_l A_{c(j,l),j} \delta_{i,c(j,l)} = \frac{1}{s} A_{ij}$$

□

## 16.4 Hermitian Block Encoding

So far we have considered general s-sparse matrices. Note that if  $A$  is Hermitian matrix, its  $(\alpha, m, \epsilon)$ -block-encoding  $U_A$  does not need to be Hermitian. Even if  $\epsilon = 0$ , we only have that the upper-left n-qubit block of  $U_A$  is Hermitian. For instance, even the block encoding of a Hermitian diagonal matrix may not be Hermitian. On the other hand, there are cases when  $U_A^\dagger = U_A$  is indeed a Hermitian matrix, and hence the definition:

**Definition 16.4.1.** (Hermitian Block Encoding). Let  $U_A$  be an  $(\alpha, m, \epsilon)$ -block-encoding of  $A$ . If  $U_A$  is also Hermitian, then it is called an  $(\alpha, m, \epsilon)$ -Hermitian-block-encoding of  $A$ . When  $\epsilon = 0$ , it is called an  $(\alpha, m)$ -Hermitian-block-encoding. The set of all  $(\alpha, m, \epsilon)$ -Hermitian-block-encoding of  $A$  is denoted by  $HBE_{\alpha,m}(A, \epsilon)$ , and we define  $HBE_{\alpha,m}(A) = HBE(A, 0)$ .

The Hermitian block encoding provides the simplest scenario of the qubitization process.

## 16.5 Query models for general sparse matrices

If we query the oracle, the assumption that for each  $l$  the value of  $c(j, l)$  is unique for all  $j$  seems unnatural for constructing general sparse matrices. So we consider an alternative method for construct the block encoding of a general sparse matrix as below.

Again without loss of generality we assume that each row/column has at most  $p = 2^s$  non zero entires, and that we have access to the following two  $(2n)$ -qubit oracles.

$$\begin{aligned} O_r |l\rangle |i\rangle &= |r(i, l)\rangle |i\rangle \\ O_c |l\rangle |j\rangle &= |c(j, l)\rangle |j\rangle \end{aligned}$$

Here  $r(i, l), c(j, l)$  gives the l-th nonzero entry in the i-th row and j-th column, respectively. It should be noted that although the index  $l \in [p]$ , we should expand it

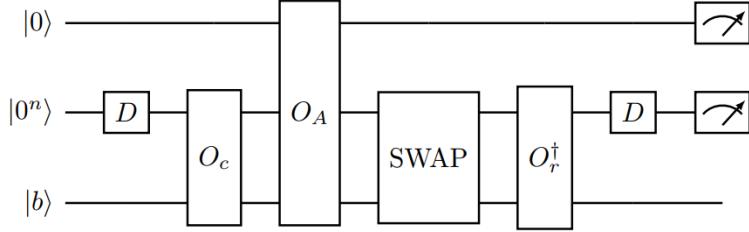


Figure 16.4: Quantum circuit for block encoding of general sparse matrices

into an  $n$ -qubit state (e.g. let  $l$  take the last  $s$  qubits of the  $n$ -qubit register following the binary representation of integers). The reason for such an expansion, and that we need two oracles  $O_c, O_r$  will be seen shortly.

Similar to the discussion before, we need a diffusion operator satisfying

$$D|0^n\rangle = \frac{1}{\sqrt{s}} \sum_{l \in [p]} |l\rangle$$

This can be implemented using Hadamard gates as

$$D = I_{n-s} \otimes H^{\otimes s}$$

We assume that the matrix entries are queried using the following oracle using controlled rotations

$$O_A|i\rangle|j\rangle = \left( A_{ij}|0\rangle + \sqrt{1 - |A_{ij}|^2}|1\rangle \right) |i\rangle|j\rangle$$

where the rotation is controlled by both row and column indices. However, if  $A_{ij} = 0$  for some  $i, j$ , the rotation can be arbitrary, as there will be no contribution due to the usage of  $O_r, O_c$ .

**Proposition 16.5.1.** *The circuit shown in figure 16.4 defines  $U_A \in BE_{s,n+1}(A)$ .*

*Proof.* We apply the first four gate sets to the source state

$$\begin{aligned} |0\rangle|0^n\rangle|j\rangle &\xrightarrow{D} \xrightarrow{O_c} \xrightarrow{O_A} \frac{1}{\sqrt{p}} \sum_{l \in [p]} \left( A_{c(j,l),j}|0\rangle + \sqrt{1 - |A_{c(j,l),j}|^2}|1\rangle \right) |c(j,l)\rangle|j\rangle \\ &\xrightarrow{SWAP} \frac{1}{\sqrt{p}} \sum_{l \in [p]} \left( A_{c(j,l),j}|0\rangle + \sqrt{1 - |A_{c(j,l),j}|^2}|1\rangle \right) |j\rangle|c(j,l)\rangle \end{aligned}$$

we then apply  $D$  and  $O_r$  to the target state

$$|0\rangle |0^n\rangle |i\rangle \xrightarrow{D} \frac{1}{\sqrt{p}} \sum_{l' \in [p]} |0\rangle |r(i, l')\rangle |i\rangle$$

Then the inner product gives

$$\begin{aligned} \langle 0 | \langle 0^n | \langle i | U_A | 0 \rangle | 0^n \rangle | j \rangle &= \frac{1}{p} \sum_{l, l'} A_{c(j, l), j} \delta_{i, c(j, l)} \delta_r(j, l'), j \\ &= \frac{1}{p} \sum_l A_{c(j, l), j} \delta_{i, c(j, l)} = \frac{1}{s} A_{ij} \end{aligned}$$

Here we have used that there exists a unique  $l$  such that  $i = c(j, l)$ , and a unique  $l'$  such that  $j = r(i, l')$ .  $\square$

We remark that the quantum circuit in figure 16.4 is essentially the construct, which gives a  $(s, n + 3)$ -block-encoding. The construct above slightly simplifies the procedure and saves two extra qubits (used to mark whether  $l \geq s$ ).

Next we consider the Hermitian block encoding of a  $s$ -sparse Hermitian matrix. Since  $A$  is Hermitian, we only need one oracle to query the location of the nonzero entries.

$$O_c |l\rangle |j\rangle = |c(j, l)\rangle |j\rangle$$

Here  $c(j, l)$  gives the  $l$ -th nonzero entry in the  $j$ -th column. It can also be interpreted as the  $l$ -th nonzero entry in the  $i$ th column. Again the first register needs to be interpreted as an  $n$ -qubit register. The diffusion operator is the same as in the equation.

Unlike all discussions before, we introduce two signal qubits and a quantum state in the computational basis take the form  $|a\rangle |i\rangle |b\rangle |j\rangle$ , where  $a, b \in \{0, 1\}$ ,  $i, j \in [N]$ . In other words, we may view  $|a\rangle |i\rangle$  as the first register, and  $|b\rangle |j\rangle$  as the second register. The  $(n + 1)$ -qubit SWAP gate is defined as

$$SWAP |a\rangle |i\rangle |b\rangle |j\rangle = |b\rangle |j\rangle |a\rangle |i\rangle$$

To query matrix entries, we need access to the square root of  $A_{ij}$  as (note that act on the second single-qubit register)

$$O_A |i\rangle |0\rangle |j\rangle = |i\rangle \left( A_{ij} |0\rangle + \sqrt{1 - |A_{ij}|} |1\rangle \right) |j\rangle$$

The square root operation is well defined if  $A_{ij} \geq 0$  for all entries. If  $A$  has negative (or complex) entries, we first write  $A_{ij} = |A_{ij}|e^{i\theta_{ij}}$ ,  $\theta_{ij} \in [0, 2\pi)$ , and the square root is uniquely defined as  $\sqrt{A_{ij}} = \sqrt{|A_{ij}|}e^{i\theta_{ij}/2}$ .

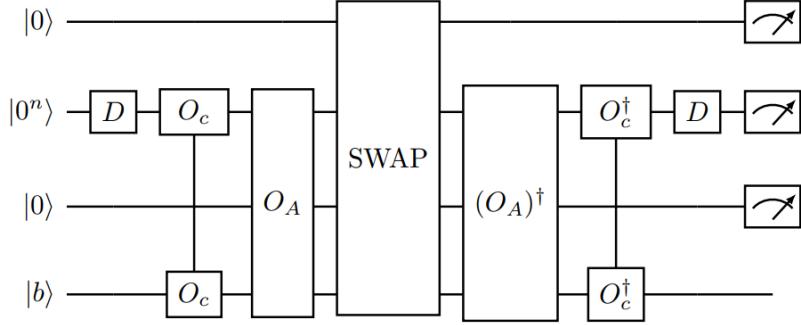


Figure 16.5: Quantum circuit for Hermitian Block encoding of a general Hermitian matrix

**Proposition 16.5.2.** *Figure 16.5 defines  $U_A \in HBE_{s,n+2}(A)$ .*

*Proof.* Apply the first four gate sets to the source state gives

$$\begin{aligned} |0\rangle |0^n\rangle |0\rangle |j\rangle &\xrightarrow{D} \xrightarrow{O_c} \xrightarrow{O_A} \frac{1}{\sqrt{s}} \sum_{l \in [s]} |0\rangle |c(j,l)\rangle (\sqrt{A_{c(j,l),j}} |0\rangle \text{sqrt1} - |A_{c(j,l),j} |1\rangle) |j\rangle \\ &\xrightarrow{\text{SWAP}} \frac{1}{\sqrt{s}} \sum_{l \in [s]} \left( \sqrt{A_{c(i,l'),i}} |0\rangle + \sqrt{1 - |A_{c(i,l'),i}|} |1\rangle \right) |i\rangle \end{aligned}$$

Apply the last three gate sets to the target state

$$|0\rangle |0^n\rangle |0\rangle |i\rangle \xrightarrow{D} \xrightarrow{O_c} \xrightarrow{O_A} \frac{1}{\sqrt{s}} \sum_{l \in [s]} |0\rangle |c(i,l')\rangle \left( \sqrt{A_{c(i,l'),i}} |0\rangle + \sqrt{1 - |A_{c(i,l'),i}|} |1\rangle \right) |i\rangle$$

Finally, take the inner product as

$$\begin{aligned} \langle 0 | \langle 0^n | \langle 0 | \langle i | U_A | 0 \rangle | 0^n \rangle | 0 \rangle | j \rangle &= \frac{1}{s} \sum_{l,l'} \sqrt{A_{c(j,l'),j}} \sqrt{A_{c(i,l'),i}^*} \delta_{i,c(j,l)} \delta_{c(i,l'),j} \\ &= \frac{1}{s} \sqrt{A_{ij} A_{ji}^*} \sum_{l,l'} \delta_{i,c(j,l)} \delta_{c(i,l'),j} = \frac{1}{s} A_{ij} \end{aligned}$$

In this equality, we have used that  $A$  is Hermitian:  $A_{ij} = A_{ji}^*$ , and there exists a unique  $l$  such that  $i = c(j, l)$ , as well as a unique  $l'$  such that  $j = c(i, l')$ .

The quantum circuit in figure 16.5 is essentially construction in. The relation will quantum walks will be discussed further.

□

# Chapter 17

## Quantum Subspace Methods

Quantum Subspace Methods (QSMs) are a class of quantum computing algorithms where the time-independent Schrodinger equation  $\hat{\mathcal{H}}\psi = E\psi$  for a quantum system is projected onto a subspace of the underlying Hilbert Space. This projection transforms the Schrodinger equation into an eigenvalue problem  $\hat{\mathcal{H}}|\psi\rangle = E|\psi\rangle$  determined by *measurements carried out on a quantum device. The eigenvalue problem is then solved on a classical computer, yielding approximations to extremal (lowest and highest) eigenvalues and their corresponding eigenvectors.* They are a hybrid quantum-classical methods, where a quantum device is supported by classical computational resources is employed to tackle a problem. The exact solution of the Schrodinger equation for a molecule with more than two elements is a formidable problem. This is because within the Born-Oppenheimer approximation, the difficulty originates from the two-electron terms, which introduce correlations in the motion of electrons under the potential generated by nuclei. For this reason, the Schrodinger equation is solved approximately.

Estimating the ground state energy of a quantum system has broad applications across physics and chemistry. The simulation of ground and excited-state properties of electronic systems is an important application of quantum computing algorithms.

**Idea: Projection of the Schrodinger equation onto a subspace of the many-electron Hilbert Space have been proposed to extract Hamiltonian Eigen pairs.** The QSMs promising class of quantum computing algorithms for near-term and fault-tolerant quantum devices. The methods studied here are the following:

1. Hamilton Krylov space
2. Quantum subroutines - real and imaginary time evolution

Then we discuss their computational cost (scaling of simulation on classical/quan-

tum computers) and accuracy (convergence properties) of classical and quantum subspace methods. Then we talk about applications of QSMs. Then implementation of QSMs on quantum devices - importance and challenges posed by finite statistical samples (shot noise) and decoherence the economisation of quantum circuits and measurements to contain the computational cost of QSMs. Finally conclusion and perspectives.

## 17.1 Basic Quantum Computing Concepts

A qubit is a physical system whose states are described by unit vectors in two dimensional Hilbert space  $\mathcal{H} = \mathbb{C}^2$ . A system of  $N_q$  qubits, also called a register, has states described by unit vectors in the Hilbert Space  $\mathcal{H}_{N_q} = \mathcal{H}^{\otimes N_q}$ . An orthonormal basis of the Hilbert space  $\mathcal{H}_{N_q}$  is given by the following vectors, called computational basis states,

$$\begin{aligned} |\mathbf{z}\rangle &= |z_{N_q-1}\rangle \otimes |z_{N_q-2}\rangle \otimes \dots \otimes |z_0\rangle = |z\rangle, z \in \{0, 1\}^{N_q}, z = \sum_{l=0}^{N_q-1} z_l 2^l \in \{0 \dots 2^{N_q-1}\} \\ &= |z_{n-1} \dots z_0\rangle \end{aligned}$$

Starting from a register of  $N_q$  qubits initialised in the state  $|0\rangle \in \mathcal{H}_{N_q}$  an  $N_q$ -qubit state  $|\Phi\rangle$  can be prepared by applying single and multi-qubit unitary transformations, or gates.

## 17.2 Single-qubit gates

Single-qubit Pauli operators

$$\mathbb{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

are very important in quantum computation. A single qubit can be prepared in a generic state  $|\Phi\rangle$  by initialisation in  $|0\rangle$  and application of single-qubit Pauli rotations, i.e. an unitary transformation of the form

$$R_\sigma(\theta) = e^{-i\frac{\theta}{2}\sigma}, \sigma \in \{X, Y, Z\}, \theta \in [0, 2\pi)$$

In the matrix form, with  $\alpha = \cos(\theta/2)$ ,  $\beta = \sin(\theta/2)$ , and  $\gamma = e^{i\theta/2}$

$$R_X(\theta) = \begin{pmatrix} \alpha & -i\beta \\ -i\beta & \alpha \end{pmatrix}, R_Y(\theta) = \begin{pmatrix} \alpha & -\beta \\ \beta & \alpha \end{pmatrix}, R_Z(\theta) = \begin{pmatrix} \gamma^* & 0 \\ 0 & \gamma \end{pmatrix}$$

It is also useful to recall that  $R_\sigma(\theta) = \cos(\theta/2)\sigma$ . A generic single qubit gate with unit determinant,  $U$ , can be written as

$$U = R_Z(\theta_2)R_Y(\theta_1)R_Z(\theta_0) = \begin{pmatrix} e^{-\frac{\theta_0+\theta_1}{2}} \cos(\theta_1/2) & -e^{\frac{\theta_0-\theta_2}{2}} \sin(\theta_1/2) \\ e^{-\frac{\theta_0-\theta_2}{2}} \sin(\theta_1/2) & e^{\frac{\theta_0+\theta_2}{2}} \cos(\theta_1/2) \end{pmatrix}$$

for suitable  $\theta_0, \theta_1, \theta_2$ , in what is called a ZYZ decomposition (one can think of these angles as equivalent Euler angles up to the double cover of  $\text{SO}(3)$  by  $\text{SU}(2)$ ).

For some superconducting qubit architectures, the available 1-qubit gates are the "square-root of X" and phase gates (the latter are applied in a virtual way by changing the phase of subsequent electromagnetic pulses, which reduces both the number of gates in the circuits and the errors, since virtual phase gates are implemented on classical software and not as physical operations) respectively.

$$\sqrt{X} = e^{i\pi/4} R_X(\pi/2) = \begin{pmatrix} \frac{1-i}{2} & \frac{1-i}{2} \\ \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix}, P(\lambda) = e^{-i\lambda/2} R_Z(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}$$

Combining  $\sqrt{X}$  and phase gates one can construct X and Y rotations, e.g.  $R_Y(\theta) = \sqrt{X}R_Z(\theta)\sqrt{X}^\dagger$ . Similarly, a generic single-qubit gate with unit determinant can be written in the following VZ decomposition, as

$$\begin{aligned} U_3(\theta, \phi, \lambda) &= \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda} \sin(\theta/2) \\ e^{i\phi} \cos(\theta/2) & e^{i(\phi+\lambda)} \cos(\theta/2) \end{pmatrix} \\ &= e^{-i\frac{\pi+\theta}{2}} P(\phi + \pi) \sqrt{X} P(\theta + \pi) \sqrt{X} P(\lambda) \end{aligned}$$

Important single-qubit gates are the Hadamard, S and T, gates, respectively.

$$Had = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, S = P(\pi/2), T = P(\pi/4)$$

## 17.3 Two-qubit gates

Single-qubit gates are not sufficient for universal quantum computation, since they cannot entangle qubits. This goal is accomplished by combining single and two-qubit gates. An important example is the canonical gate

$$Can(t_x, t_y, t_z) = e^{-i\frac{\pi}{2}(t_x X \otimes X + t_y Y \otimes Y + t_z Z \otimes Z)} = R_{xx}(\pi t_x) R_{yy}(\pi t_y) R_{zz}(\pi t_z),$$

where the parameters  $t_x$

## 17.4 Classical Subspace Methods

Subspace methods construct finite dimensional approximation to the eigenvalue equation  $\hat{\mathcal{H}}|\psi\rangle = E|\psi\rangle$ , an approach called Galerkin's method. Specifically, given a set of many electron wave functions  $|v_0\rangle, \dots, |v_{n-1}\rangle$ , we seek a function of the form  $|\psi_\mu\rangle = \sum_{\beta=0}^{n-1} C_{\beta\mu} |v_\beta\rangle$ , that solves the projection equation of the Schrodinger equation  $\hat{\mathcal{H}}|\psi\rangle = E|\psi\rangle$  onto the subspace  $V_n = \text{Span}(|v_0\rangle, \dots, |v_{n-1}\rangle)$ , i.e., for each  $\alpha, \mu \in [n] := \{0, 1, \dots, n-1\}$

$$\sum_{\beta=0}^{n-1} \hat{H} C_{\beta\mu} |v_\beta\rangle = E_\mu \sum_{\beta=0}^{n-1} C_{\beta\mu} |v_\beta\rangle$$

Now taking inner product with  $\langle v_\alpha |$ , we get,

$$\sum_{\beta=0}^{n-1} \langle v_\alpha | \hat{H} | v_\beta \rangle C_{\beta\mu} = E_\mu \sum_{\beta=0}^{n-1} \langle v_\alpha | v_\beta \rangle C_{\beta\mu}$$

for  $\alpha, \mu \in [n]$ . In the matrix form this can be written as follows:

$$\begin{bmatrix} \langle v_0 | \hat{H} | v_0 \rangle & \langle v_0 | \hat{H} | v_1 \rangle & \dots & \langle v_0 | \hat{H} | v_{n-1} \rangle \\ \langle v_1 | \hat{H} | v_0 \rangle & \langle v_1 | \hat{H} | v_1 \rangle & \dots & \langle v_1 | \hat{H} | v_{n-1} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle v_{n-1} | \hat{H} | v_0 \rangle & \langle v_{n-1} | \hat{H} | v_1 \rangle & \dots & \langle v_{n-1} | \hat{H} | v_{n-1} \rangle \end{bmatrix} \begin{bmatrix} C_{00} & C_{01} & \dots & C_{0n-1} \\ C_{10} & C_{11} & \dots & C_{1n-1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{n-10} & C_{n-11} & \dots & C_{n-1n-1} \end{bmatrix} =$$
  

$$\begin{bmatrix} \langle v_0 | v_0 \rangle & \langle v_0 | v_1 \rangle & \dots & \langle v_0 | v_{n-1} \rangle \\ \langle v_0 | v_0 \rangle & \langle v_0 | v_1 \rangle & \dots & \langle v_0 | v_{n-1} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle v_{n-1} | v_0 \rangle & \langle v_{n-1} | v_1 \rangle & \dots & \langle v_{n-1} | v_{n-1} \rangle \end{bmatrix} \begin{bmatrix} C_{00} & C_{01} & \dots & C_{0n-1} \\ C_{10} & C_{11} & \dots & C_{1n-1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{n-10} & C_{n-11} & \dots & C_{n-1n-1} \end{bmatrix} \begin{bmatrix} E_0 & 0 & \dots & 0 \\ 0 & E_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & E_{n-1} \end{bmatrix}$$

This is called **Generalised Eigenvalue Equation (GEEV)** which can be compactly written as  $HC = SCE$ , where  $H$  and  $S$  denote the Hamiltonian and overlap matrices, respectively, whose entries are

$$H_{\alpha\beta} = \langle v_\alpha | \hat{H} | v_\beta \rangle : \quad S_{\alpha\beta} = \langle v_\alpha | v_\beta \rangle$$

### Important Note

The approximate Eigenvalue  $E_\mu$  and eigen vectors  $|\psi_\mu\rangle$  differ from the exact ones since the approximate eigenvectors are restricted to subspace  $V_n$ . The quality of a subspace method depends on several factors:

1. the nature of the basis vectors  $|v_\beta\rangle$
2. the dimension  $n$  of the employed subspace
3. the cost (and associated error) of computing the matrices  $H, S$
4. the numerical stability of the equation  $HC = SCE$ ., connected with the condition number of the overlap matrix  $S$ .

Classical Subspace methods include:

1. Krylov Space Methods
2. Lanczos Method

### Block Encoding

**Definition 17.4.1.** A block encoding of a Hamiltonian  $H$  (acting on a Hilbert space  $\mathcal{H}_f$  whose states are denoted  $|\cdot\rangle_s$ ) is a pair  $(U, G)$ , where  $U$  is unitary acting on  $\mathcal{H}_a \otimes \mathcal{H}_s$  (for some auxiliary Hilbert space  $\mathcal{H}_a$  whose states are denoted  $|\cdot\rangle_a$ ), and  $|G\rangle_a = G|0\rangle_a$  is a state of the auxiliary qubits such that

$$(\langle G|_a \otimes I_s)U(|G\rangle_a \otimes I_s) = H$$

where  $I_s$  denotes identity action on  $\mathcal{H}_s$ .

Consider the following example of block encoding of an  $n$ -qubit Hamiltonian.

**Example 17.4.1.** A Hamiltonian can be expressed as a linear combination of Pauli operators.

$$H = \sum_{i=0}^{T-1} \alpha_i P_i$$

where  $\alpha_i$  are real coefficients,  $P_i$  are Pauli operators, and the number of terms is  $T = O(\text{poly}(n))$ . We assume that the coefficients  $\alpha_i$  are non-negative, and

instead each Pauli  $P_i$  carries a  $\pm 1$  sign. We also require a normalization condition on  $H : \sum_{i=0}^{T-1} |\alpha_i| = \sum_{i=0}^{T-1} \alpha_i = 1$ . Given some non-normalized input Hamiltonian whose coefficients are  $\alpha_i^{(E)}$ , we obtain normalized coefficients as  $\alpha_i = |\alpha_i^{(E)}| / \sum_{i=0}^{T-1} |\alpha_i^{(E)}|$ . The block encoding unitary is

$$U = \sum_{i=0}^{T-1} |i\rangle_a \langle i|_a \otimes P_i$$

i.e., application of each Pauli term  $P_i$  controlled on the state of the auxiliary register being  $|i\rangle_a$ . The corresponding block encoding state is

$$|G\rangle_a = \sum_{i=0}^{T-1} \sqrt{\alpha_i} |i\rangle_a$$

Inserting  $U$  as defined and  $|G\rangle_a$  and using the fact that  $\alpha_i \geq 0$  for all  $i$ , one can verify that

$$\left( \sum_{i=0}^{T-1} \sqrt{\alpha_i} |i\rangle_a \otimes I_s \right) \left( \sum_{j=0}^{T-1} |j\rangle_a \langle j|_a \otimes P_j \right) \left( \sum_{k=0}^{T-1} \sqrt{\alpha_k} |k\rangle_a \otimes I_s \right)$$

which gets simplified to

$$\begin{aligned} & \sum_{i=0}^{T-1} \sum_{j=0}^{T-1} \sum_{k=0}^{T-1} \sqrt{\alpha_i} \sqrt{\alpha_k} \langle i|j\rangle_a \langle j|k\rangle_a \otimes P_j \\ & \sum_{i=0}^{T-1} \sum_{j=0}^{T-1} \sum_{k=0}^{T-1} \sqrt{\alpha_i} \sqrt{\alpha_k} \delta_{ij} \delta_{jk} \otimes P_j \\ & \sum_{i=0}^{T-1} \alpha_i \otimes P_i = H \end{aligned}$$

Thus,  $(U, G)$  forms a valid block encoding of  $H$ .

### 3. Davidson Method

### 17.4.1 Krylov Space Methods

### 17.4.2 Lanczos Method

## 17.5 Quantum Subspace Methods

Classically, estimating the ground state energy of a quantum system is believed to be exponentially hard in the system size, due to the cost of representing vectors in the Hilbert space. Many algorithms for the task have been proposed that avoid this representation problem by the encoding the system's Hilbert space in the Hilbert space of a register of qubits, whose dimension also grows exponentially. The problem of finding ground state energies of local Hamiltonians is **QMA-Hard**. Thus, sufficiently generally quantum algorithms for approximating ground state energies must require additional assumptions. Some of the typical assumptions are finding an initial guess state that has  $\Omega(1/\text{poly}(n))$  overlap with sufficiently low-energy states (for  $n$  qubits), finding a gapped adiabatic path from an efficiently solvable Hamiltonian to the target Hamiltonian, optimizing variational quantum circuit, or assuming bounded correlation strength.

### 17.5.1 Idea of the Algorithm

The quantum subspace expansion (QSE) method constructs a variational subspace spanned by a set of states  $\{|v_\beta\rangle\}_{\beta=0}^{n-1}$ . Hamiltonian eigenstates are approximated by a linear combination of basis states,  $|\psi_\mu\rangle = \sum_\beta C_{\beta\mu} |v_\beta\rangle$ . To obtain expansion coefficients  $C_{\beta\mu}$ , one computes matrix elements  $S_{\alpha\beta} = \langle v_\alpha | v_\beta \rangle$  and  $H_{\alpha\beta} = \langle v_\alpha | \hat{H} | v_\beta \rangle$  in the chosen basis and solves the GEEV  $HC = SCE$ . Within QSE, a quantum device is used to compute the matrix elements  $H_{\alpha\beta}$  and  $S_{\alpha\beta}$ , and a classical computer to solve the GEEV and obtain approximate eigenpairs  $\{E_\mu, |\psi_\mu\rangle\}$ . The quantum circuits required to compute  $H_{\alpha\beta}$  and  $S_{\alpha\beta}$  depend on the choice of the basis  $|v_\alpha\rangle_{\alpha=0}^{n-1}$ .

### 17.5.2 Difference between QSE and VQE

The main advantage of QSE (and other subspace methods) over the well-established variational quantum eigensolver (VQE) is that QSE does not require a non-linear parameter optimisation, which is an NP-hard problem, as part of the eigen value approximation. While both QSE and VQE employ quantum and classical computers in synergy, the nature of their interaction is profoundly different: in VQE, each update of the quantum circuit parameters requires a new call of the quantum computer in

just one call. Furthermore, QSE circuits can be naturally parallelised over multiple quantum computers (or groups of qubits within a quantum computer). A limitation of QSE is that the eigen states  $|\psi_\mu\rangle$  are never actually stored on a quantum device, so computing properties after a QSE calculation generally requires additional measurements.

The accuracy and computational cost of a QSE calculation depends on the choice of the basis states  $\{|v_\alpha\rangle\}_{\alpha=0}^{n-1}$ . However, no specific prescription is provided for the selection, similar to the original Galerkin's method. In this sense, QSE can be regarded as family of QSMs, differentiated by the choice of the basis states.

### 17.5.3 Hamiltonian Krylov Space Methods

Note that constructing a Krylov space generated by powers of the Hamiltonian on a quantum computer is non-trivial. This is because the Hamiltonian and its powers are non-unitary operations, thus cannot be trivially mapped onto a quantum circuit. The use of block-encoding unitaries was proposed to circumvent this limitation.

#### Chebysev Quantum Krylov

It is Quantum implementation of the classical Krylov space. This method is exact in the sense that it has no algorithmic error in the construction of the Krylov space i.e. approximations such as Trotterization or other approximate algorithms are not required.

# Chapter 18

## Variational Algorithms

To write Variational algorithms:near-term hybrid quantum-classical algorithms that are ideal candidates to achieve quantum advantage. The aim of this section is to learn:

- each step in variational algorithm design workflow
- trade offs associated with each step
- how to use Qiskit Runtime primitives to optimize for speed and accuracy.

In this chapter we will see the specifics of variational algorithms and near-term hybrid quantum-classical algorithms based on the variational theorem of quantum mechanics. These algorithms can leverage the utility provided by today's non-fault-tolerant quantum computers, making them ideal candidates to achieve quantum advantage.

Variational algorithms include several modular components that can be combined and optimized based on algorithm, software and hardware advancements. This includes a cost function that describes a specific problem with a set of parameters, an ansatz to express the search space with these parameters, and an optimizer to iteratively explore the search space. During each iterations, the optimizer evaluates the cost function with the current parameters and selects the next iteration's parameters until it converges on an optimal solution. The hybrid nature of this family of algorithms comes from the fact that the cost functions are evaluated using quantum resources and optimized through classical ones.

1. **Initialize Problem:** Variational algorithms start by initializing the quantum computer in a default state  $|0\rangle$ , then transforming it to some desired (non-parameterized) state  $|\rho\rangle$ , which we will call *reference state*.



Figure 18.1: Variational Algorithm

This transformation is represented by the application of a unitary reference operator  $U_R$  on the default state, such that  $U_R |0\rangle = |\rho\rangle$ .

2. **Prepare ansatz:** To begin iteratively optimizing from the default state  $|0\rangle$  to the target state  $|\psi(\vec{\theta})\rangle$ , we must define a *variational form*  $U_V(\vec{\theta})$  to represent a collection of parameterized states for our variational algorithm to explore. We refer to any particular combination of reference state and variational form as an ansatz state such that  $U_A(\vec{\theta}) := U_V(\vec{\theta})U_R$ . Ansätze will ultimately take the form of parameterize quantum circuits capable of taking the default state  $|0\rangle$  to the target state  $|\psi(\vec{\theta})\rangle$ .

All in all we will have:

$$\begin{aligned} |0\rangle &\xrightarrow{U_R} U_R |0\rangle = |\rho\rangle \xrightarrow{U_V(\vec{\theta})} U_A(\vec{\theta}) |0\rangle \\ &= U_V(\vec{\theta})U_R |0\rangle \\ &= U_V(\vec{\theta}) |\rho\rangle \\ &= |\psi(\vec{\theta})\rangle \end{aligned}$$

3. **Evaluate Cost function:** We can encode our problem into a *cost function*  $C(\vec{\theta})$  as a linear combination of Pauli operators , run on a quantum system. While this can be information about a physical system, such as energy or spin, we can also encode non-physical problems as well. We can leverage Qiskit Runtime primitives to address noise with error suppression and mitigation while evaluating our cost function.
4. **Optimize Parameters:** Evaluations are taken to a classical computer, where a classical optimizer analyzes them and chooses the next set of values for the variational parameters. If we have a pre-existing optimal solution, we can set it as an *initial point*  $\vec{\theta}_0$  to bootstrap our optimization. Using this *initial state*  $|\psi(\vec{\theta}_0)\rangle$  could help our optimizer find a valid solution faster.
5. **Adjust ansatz parameters with results, and re-run:** the entire process is repeated until the classical optimizer's finalization criteria are met, and an optimal set of parameter values  $\vec{\theta}^*$  is returned. the proposed solution state for our problem will be  $|\psi(\vec{\theta}^*)\rangle = U_A(\vec{\theta}^*) |0\rangle$ .

## 18.1 Variational Theorem

A common goal of variational algorithms is to find the quantum state with the lowest or highest eigenvalue of a certain observable. A key insight we'll use is the variation theorem of quantum mechanics. before going into its full statement, let us explore some of the mathematical intuition behind it.

### 18.1.1 Mathematical intuition for energy and ground state

In quantum mechanics, energy comes in the form of a quantum observable usually referred to as the *Hamiltonian*, which we'll denote by  $\hat{\mathcal{H}}$ . let us consider its spectral decomposition:

$$\hat{\mathcal{H}} = \sum_{k=0}^{N-1} \lambda_k |\phi_k\rangle \langle \phi_k|$$

where  $N$  is the dimensionality of the space of state,  $\lambda_k$  is the k-th eigenvalue or, physically, the k-th energy level, and  $|\phi_k\rangle$  is the corresponding eigen state:  $\hat{\mathcal{H}}|\phi_k\rangle = \lambda_k|\phi_k\rangle$ , the expected energy of a system in the (normalized) state *ket* $\psi$  will be:

$$\begin{aligned} \langle\psi|\hat{\mathcal{H}}|\psi\rangle &= \langle\psi| \left( \sum_{k=0}^{N-1} \lambda_k |\phi_k\rangle \langle k| \right) |\psi\rangle \\ &= \sum_{k=0}^{N-1} \lambda_k \langle\psi|\phi_k\rangle \langle\phi_k|\psi\rangle \\ &= \sum_{k=0}^{N-1} -k = 0^{N-1} \lambda_0 |\langle\psi|\phi_0\rangle|^2 \end{aligned}$$

If we take into account that  $\lambda_0 \leq \lambda_k, \forall k$ , we have:

$$\begin{aligned} \langle\psi|\hat{\mathcal{H}}|\psi\rangle &= \sum_{k=0}^{N-1} |\langle\psi|\phi_k\rangle|^2 \\ &\geq \sum_{k=0}^{N-1} \lambda_0 |\langle\psi|\phi_k\rangle|^2 \\ &= \lambda_0 \sum_{k=0}^{N-1} |\langle\psi|\phi_k\rangle|^2 \\ &= \lambda_0 \end{aligned}$$

Since  $\{|\phi_k\rangle\}_{k=0}^{N-1}$  is an orthonormal basis, the probability of measuring  $|\phi_k\rangle$  is  $p_k = |\langle\psi|\phi_k\rangle|^2$ , and the sum of all the probabilities is such that  $\sum_{k=0}^{N-1} |\langle\psi|\phi_k\rangle|^2 = \sum_{k=0}^{N-1} p_k = 1$ . In short, the expected energy of any system is higher than the lowest energy or ground state energy:

$$\langle\psi|\hat{\mathcal{H}}\psi\rangle \geq \lambda_0$$

the above argument applies to any valid (normalized) quantum state  $|\psi\rangle$ , so it is perfectly possible to consider parameterized state  $|\psi(\vec{\theta})\rangle$  that depends on a parameter vector  $\vec{\theta}$ . This is where the "variational" part comes into play. If we consider a cost function given by  $C(\vec{\theta}) := \langle\psi(\vec{\theta})|\hat{\mathcal{H}}|\psi(\vec{\theta})\rangle$  and we want to minimize it, the minimum will always satisfy:

$$\min_{\vec{\theta}} C(\vec{\theta}) = \min_{\vec{\theta}} \langle\psi(\vec{\theta})|\hat{\mathcal{H}}|\psi(\vec{\theta})\rangle \geq \lambda_0$$

The minimum value of  $C(\vec{\theta})$  will be the closest that one can get to  $\lambda_0$  using the parameterized state  $|\psi(\vec{\theta})\rangle$ , and equality will only be reached if there exists a parameter vector  $\vec{\theta}^*$  such that  $|\vec{\theta}^*\rangle = |\phi_0\rangle$

### 18.1.2 Variational theorem of Quantum Mechanics

If the (normalized) state  $|\psi\rangle$  of a quantum system depends on a parameter vector  $\vec{\theta}$ , then the optimal approximation of the ground state (i.e. the eigen state  $|\phi_0\rangle$  with the minimum value  $\lambda_0$ ) is the one that minimizes the expectation value of the Hamiltonian  $\hat{\mathcal{H}}$ :

$$\langle\hat{\mathcal{H}}\rangle(\vec{\theta}) := \langle\psi(\vec{\theta})|\hat{\mathcal{H}}|\psi(\vec{\theta})\rangle \geq \lambda_0$$

The reason why the variational theorem is stated in terms of energy minima is that it includes a number of mathematical assumptions:

- For physical reasons, a finite lower bound to the energy  $E \geq \lambda_0 > -\infty$  needs to exist, even for  $N \rightarrow \infty$ .
- Upper bounds do not generally exist.

However, mathematically speaking, there is nothing special about the Hamiltonian  $\hat{\mathcal{H}}$  beyond these assumptions, so the theorem can be generalized to other quantum observables and their eigenstates provided they follow the same constraints. Also, note that if finite upperbounds exist, the same mathematical arguments could be made for maximizing eigen values by swapping lower bounds for upper bounds.

## 18.2 Reference States

In this section, we will explore how we can initialize our system with a reference state to help our variational algorithm converge faster. First, we will learn how to construct a reference state manually, and then explore several standard options that can be used in a variational algorithm.

### 18.2.1 Default State

A *reference state* refers to the initial fixed part of our problem. To prepare a reference state, we need to apply the appropriate, non-parameterized unitary  $U_R$  at the start of our quantum circuit, such that  $|\rho\rangle = U_R|0\rangle$ . If you have an educated guess or data point from an existing optimal solution, the variational algorithm will likely converge faster if you use that as a starting point.

The simplest possible reference state is the default state, where we use the starting state of an  $n$ -qubit quantum circuit:  $|0^{\otimes n}\rangle$ . For the default state, our unitary operator  $U_R = I$ . Due to its simplicity, the default state is a valid reference state used in many scenarios.

### 18.2.2 Classical Reference state

Suppose you have a 3-qubit system and you want to start in the state  $|001\rangle$  instead of the default state  $|000\rangle$ . This is an example of a purely classical reference state, and to construct it, you simply need to apply an X gate to qubit 0 (following Qiskit's qubit ordering), as  $|001\rangle = X_0|000\rangle$ .

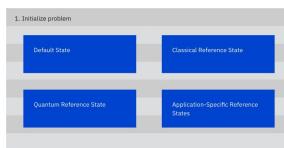


Figure 18.2: Reference state

In this case, our unitary operator  $U_R = X_0$ , which leads to the reference state  $|\rho\rangle = |001\rangle$ .

```

1 from qiskit import QuantumCircuit
2 qc=QuantumCircuit(3)
3 qc.x(0)
4

```

```
5 qc.draw("mpl")
```

This outputs the following circuit:

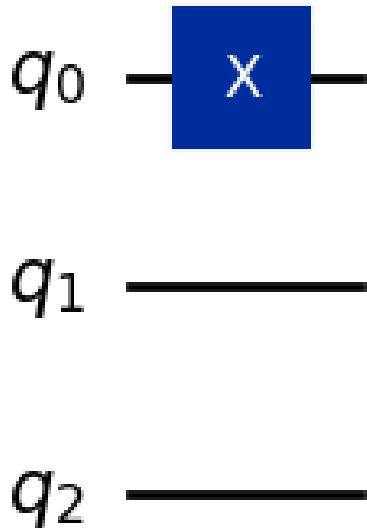


Figure 18.3: Classical Reference State Circuit

### 18.2.3 Quantum Reference State

Suppose you aim to start with a more complex state that involves superposition and/or entanglement such as  $\frac{1}{\sqrt{2}}(|100\rangle + |111\rangle)$ .

To obtain this state from  $|ket000\rangle$ , one approach is to use a hadmadrdrd gate on qubit 0( $H_0$ ), a CNOTE(CX) gate with qubit 0 as the control qubit and qubit 1 as the target qubit ( $CNOT_{01}$ ), and finally an X gate applied to qubit 2 ( $X_2$ ).

In this scenario, our unitary operator is  $U_R = X_2CNOT_{01}H_0|000\rangle$ , and our reference state is  $|\rho\rangle = \frac{1}{\sqrt{2}}(|100\rangle + |111\rangle)$ .

```
1 qc=QuantumCircuit(3)
2 qc.h(0)
3 qc.cx(0,1)
```

```

4 qc.x(2)
5
6 qc.draw("mpl")

```

This outputs the following circuit:

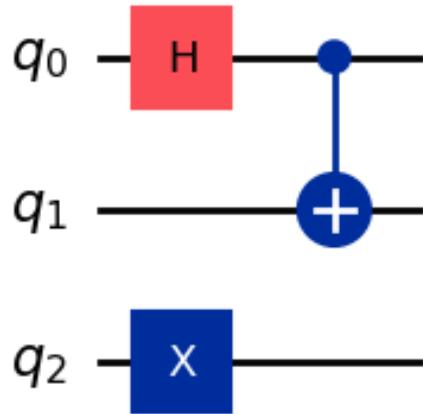


Figure 18.4: Quantum Reference State

#### 18.2.4 Constructing Reference States using template circuits

We can also use various template circuits, such as `TwoLocal` which allows for expressing multiple tunable parameters and entanglements with ease. We will cover these template circuits in more detail in the next section, but we can use them fr our reference state if we bind the parameters.

```

1 from qiskit.circuit.library import TwoLocal
2 from math import pi
3
4 reference_circuit = twoLocal(2,"rx","cz",entanglement="linear",reps=1)
5 theta_list = [pi//2,pi/2,pi/2,pi/2]
6
7 reference_circuit = reference_circuit.assign_parameters(theta_list)
8
9 reference_circuit.decompose().draw("mpl")

```

This outputs the following circuit:

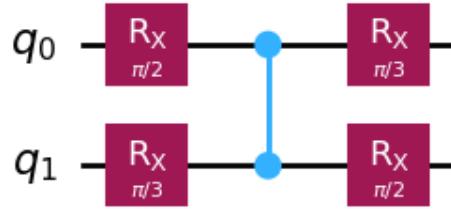


Figure 18.5: Reference State using Template

### 18.2.5 Application Specific Reference States

#### Quantum Machine Learning

In the context of a variational quantum classifier (VQC), the training data is encoded into a quantum state with a parameterized circuit known as a feature map, where each parameter value represents a data point from the training dataset. The ZZFeatureMap is a type of parameterized circuit that can be utilized to pass our data point(x) to this feature map.

```

1 from qiskit.circuit.library import ZZFeatureMap
2 data=[0.1,0.2]
3
4 zz_feature_map_reference = ZZFeatureMap(feature_dimension=2, reps=2)
5 zz_feature_map_reference = zz_feature_map_reference.assign_parameters(
6     data)
6 zz_feature_map_reference.decompose().draw("mpl")
    
```

This outputs the following circuit:

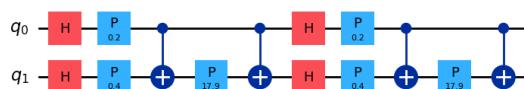


Figure 18.6: Reference State using ZZFeatureMap

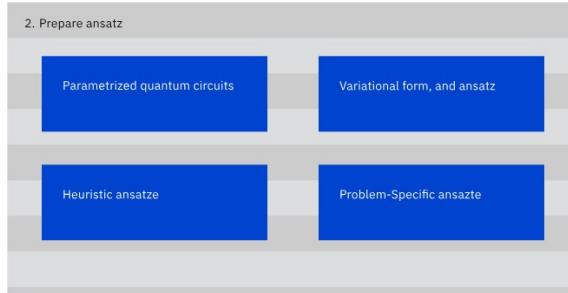


Figure 18.7: Prepare Ansatz

## 18.3 Ansatze and Variational Forms

At the heart of all variational algorithm lies the key idea of analyzing the differences between states, which are conveniently related through some well-behaved mapping (e.g., continuous, differentiable) from a set of parameters or variables - hence the name.

First, we'll explore how to construct parameterized circuits by hand. We'll use these circuits to define a variational form that represents a collection of parameterized states for our variational algorithms to explore. Then, we'll construct our ansatz by applying this variational form to our reference state.

We'll also explore how to trade off speed versus accuracy while exploring this search space.

### 18.3.1 Parameterized Quantum Circuits

Variational algorithms operate by exploring and comparing a range of quantum state  $|\psi(\vec{\theta})\rangle$ , which depends on a finite set of  $k$  parameters  $\vec{\theta} = (\theta^0, \dots, \theta^{k-1})$ . These states can be prepared using a parameterized quantum circuit, where gates are defined with tunable parameters. It is possible to create this parameterized circuit without binding specific angles yet:

```

1 from qiskit.circuit import QuantumCircuit, Parameter
2
3 theta = Parameter("theta")
4
5 qc = QuantumCircuit(3)
6 qc.rx(theta, 0)
7 qc.cx(0, 1)
8 qc.x(2)
9

```

```
10 qc.draw("mpl")
```

This outputs the following parameterized circuit:

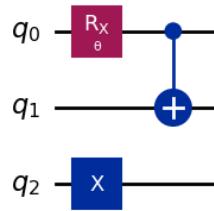


Figure 18.8: Parameterized Quantum Circuit

```
1 from math import pi
2
3 angle_list=[pi/3, pi/2]
4 circuits = [qc.assign_parameters({theta:angle} from angle in angle_list
5 )
6 for circuit in circuits:
7     display(circuit.draw("mpl"))
```

This outputs the following circuits:

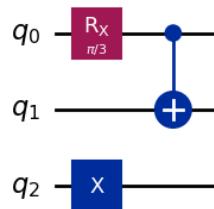
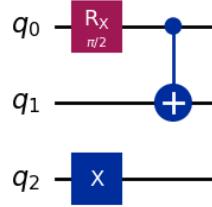


Figure 18.9: For  $\theta = \pi/3$

Figure 18.10: For  $\theta = \pi/2$ 

### 18.3.2 Variational Form and Ansatz

To iteratively optimize from a reference state  $|\rho\rangle$  to a target state  $|\psi(\vec{\theta})\rangle$ , we need to define a variational form  $U_V(\vec{\theta})$  that represents a collection of parameterized state for our variational algorithm to explore:

$$\begin{aligned} |0\rangle &\xrightarrow{U_R} U_R |0\rangle = |\rho\rangle \xrightarrow{U_V(\vec{\theta})} U_A(\vec{\theta}) |0\rangle \\ &= U_V(\vec{\theta}) U_R |0\rangle \\ &= U_V(\vec{\theta}) |\rho\rangle \\ &= |\psi(\vec{\theta})\rangle \end{aligned}$$

Note that the parameterized state depends on both the reference state  $|\rho\rangle$ , which does not depend on any parameters, and the variational form  $U_V(\vec{\theta})$ , which always depends on parameters. We refer to the combination of these two halves as an ansatz:  $U_A(\vec{\theta}) := U_V(\vec{\theta}) U_R$ .

As we construct our ansatz to represent a collection of parameterized states for our variational algorithm to explore, we realize an important issue: dimensionality. An  $n$ -qubit system (i.e., Hilbert space) has a vast number of distinct quantum states in the configuration space. We would require an unwieldy number of parameters to fully explore it. Quantitatively, its dimensionality is  $D = 2^{2n}$ . To make matters worse, the runtime complexity of search algorithms, and others alike, grows exponentially with this dimensionality, a phenomenon often referred to in the literature as the curse of dimensionality.

To counter this setback, it is common practice to impose some reasonable constraints on the variational form such that only the most relevant states are explored. Finding an efficient truncated ansatz is an active area of research, but we'll cover two common designs.

### Heuristic ansatze and trade-offs

If you do not have any information about your particular problem that can help restrict the dimensionality, you can try any arbitrary family of parameterized circuits with fewer than  $2^{2n}$ . However, there are some trade-offs to consider:

- **Speed:** By reducing the search space, the algorithm can run faster.
- **Accuracy:** However, reducing the space could risk excluding the actual solution to the problem, leading to suboptimal solutions.
- **Noise:** Deeper circuits are affected by noise, so we need to experiment with our ansatz's connectivity, gates and gate fidelity.

There is a fundamental trade-off between quality (or even solvability) and speed: the more parameters, the more likely you are to find a precise result, but the longer it will take to run the algorithm.

### N-local circuits

One of the most widely used examples of heuristic ansatzes is the N-local circuits, for a few reasons:

- **Efficient Implementation:** The N-local ansatz is typically composed of simple, local gates that can be implemented efficiently on a quantum computer, using a small number of physical qubits. This makes it easier to construct and optimize quantum circuits.
- **Captures important correlations:** The N-local ansatz can capture important correlations between the qubits in a quantum system, even with a small number of gates. This is because the local gates can act on neighboring qubits and create entanglement between them, which can be important for simulating complex quantum systems.

These circuits consist of rotation and entanglement layers that are repeated alternatively one or more times as follows:

- Each layer is formed by gates of size at most N, where N has to be lower than the number of qubits.
- For a rotation layer, the gates are stacked on top of each other. We can use standard rotation operations such as RX or CRZ. For an entanglement layer, we can use gates like Toffoli gates or CS with an entanglement strategy.

- Both types of layers can be parameterized or not, but at least one of them needs to contain parameters. Otherwise, without at least one parameter, there wouldn't be any variations!
- Optionally, an extra rotation layer can be added to the end of the circuit.

For example, let's create a 5-qubit Nlocal circuit with rotation blocks formed by RX and CRZ gates, entanglement blocks formed by Toffoli gates that act on n qubits  $[0, 1, 2], [0, 2, 3], [4, 2, 1]$  and  $[3, 1, 0]$  and 2 repetitions of each layer.

```

1 from qiskit.circuit.library import NLocal, CCXGate, CRZgate, RXGate
2 from qiskit.circuit import Parameter
3
4 theta=Parameter("theta")
5 ansatz=NLocal(num_qubits=5,rotation_blocks=[RXGate(theta),
     CRZGate9theta)], entanglement_blocks=CCXGate(),entanglement
     =[[0,1,2],[0,2,3],[4,2,1],[3,1,0]], reps=2,insert_barriers=True)
6 ansatz.decompose().draw("mpl")

```

This results in the following circuit:

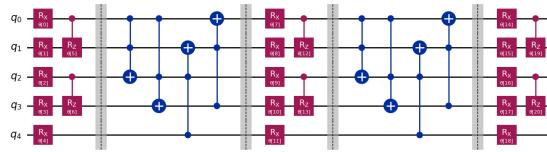


Figure 18.11: NLocal 5qubit example Circuit

In the above example, the largest gate is the Toffoli gate, which acts on three qubits, making the circuit 3-local. The most commonly used type of N-local circuits are 2-local circuits with single-qubit rotation gates and 2-qubit entanglement gates.

Let's create a 2-local circuit using Qiskit's TwoLocal class. The syntax is the same as NLocal, but there are some differences. For instance, most gates, such as RX, RZ, and CNOT, can be passed as strings without importing the gates or creating a Parameter instance.

```

1 from qiskit.circuit.library import TwoLocal
2
3 ansatz=TwoLocal(numqubits=5, rotation_blocks=["rx","rz"]),
     entanglement_blocks="cx",entanglement="linear",reps=2,
     insert_barriers=True,)
4 ansatz.decompose().draw("mpl")

```

The circuit will be as shown:

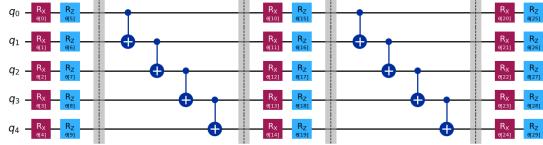


Figure 18.12: TwoLocal Ansatz Circuit Example

In this case, we used the linear entanglement distribution, where each qubit is entangled with the next. to learn about other strategies, refer to Twolocal's documentation.

## EfficientSU2

EfficientSU2 is a hardware-efficient circuit that consists of single-qubit operations spanning SU(2) and CX entanglements. This is a heuristic pattern that can be used to prepare trial wave functions for variational quantum algorithms or as a classification circuit for machine learning.

```

1 from qiskit.circuit.library import EfficientSU2
2
3 ansatz=EfficientSU2(4,su2_gates=["rx","y"],entanglement="linear",reps
4 =1)
5 ansatz.decompose().draw("mpl")

```

This results in following circuit:

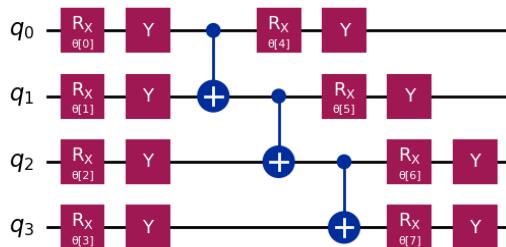


Figure 18.13: EfficientSU2 ansatze

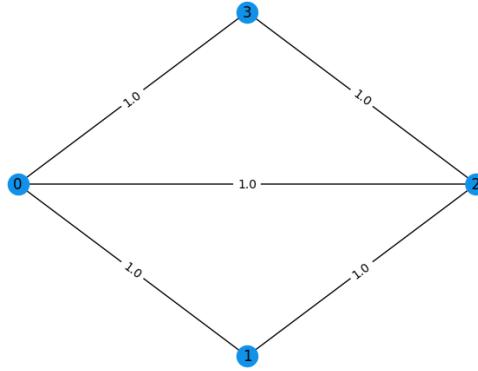


Figure 18.14: graph

### 18.3.3 Problem-specific ansatze

While heuristic and hardware efficient ansatze help us to solve a problem in a naive way, we can cause problem-specific knowledge to restrict our circuit search space to a specific type. This will help us to gain speed without losing accuracy in our search process.

#### Optimization

In a max-cut problem, we want to partition nodes of a graph in a way that maximizes the number of edges between nodes in different groups. The desired max-cut partition for the graph below is clear: the 0th node on the left should be separated from the rest of the nodes in the right by a cut.

```

1 import rustworkx as rx
2 from rustworkx.visualization import mpl_draw
3
4 n=4
5 G=rx.PyGraph()
6 G.add_nodes_from(range(n))
7 # The edge syntax is (start , end , weight)
8 edges=[(0,1,1.0),(0,2,1.0),(0,3,1.0),(1,2,1.0),(2,3,1.0)]
9 G.add_edges_from(edges)
10
11 mpl_draw(G, pos=rx.shell_layout(G), with_labels=True, edge_labels=str,
           node_color="#1192E8")
  
```

The following is the resulting graph:

To utilize QAOA algorithm from a max-cut problem, we require a Pauli-Hamiltonian that encodes the cost in a manner such that the minimum expectation value of the operator corresponds to the maximum number of edges between the nodes in two different groups.

For this, simple example, the operator is a linear combination of terms with Z operators on nodes connected by an edge (recall that the 0th qubit is farthest right): ZZII+IZZI+ ZIZZ+IZIZ+IIZZ. Once the operator is constructed, the ansatz for the QAOA algorithm can easily be built by using the QAOA Ansatz circuit from the Qiskit circuit library.

```

1 # Pre-defined ansatz circuit , operator class and visualization too:
2 from qiskit.circuit.library import QAOAAnsatz
3 from qiskit.quantum_info import SparsePauliOp
4
5 # Problem to Hamiltonian operator
6 hamiltonian = SparsePauliOp.from_list([("ZZII",1), ("IZZI",1), ("ZIIZ",1),
7                                         ("IZIZ",1), ("IIZZ",1)])
7 # QAOA ansatz circuit
8 ansatz=QAOAAnsatz(hamiltonian, reps=2)
9 # Draw
10 ansatz.decompose(reps=3).draw("mpl")

```

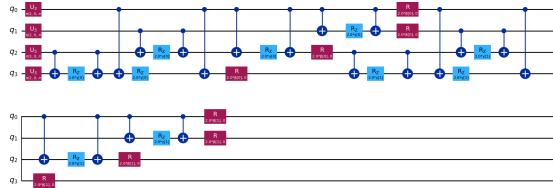


Figure 18.15: QAOA circuit

The previous image illustrates the ansatz in basic gates for clarity. However, it can be expressed in multiple levels of decomposition by changing the reps argument or by drawing the circuit without the decompose method. for example, the following representation directly shows the QAOA structure with the default reps value, which is reps=1.

```
1 ansatz.decompose(reps=1).draw("mpl")
```



Figure 18.16: Decompose Circuit

## Quantum Machine Learning

In machine learning, a common application is the classification of data into two or more categories. This involves encoding a datapoint into a feature map that maps classical feature vectors into the quantum hilbert space. Constructing quantum feature maps based on the parameterized quantum circuits. Constructing quantum feature meaps based on parameterized quantum circuits that are hard to simulate classically is an important step towards obtaining a potential adantage over classical machine learning approached and is an active are of current research.

The ZZFeatureMap can be used to create a parameterized circuit. We can pass in our data points to the feature map ( $x$ ) and a separate variational form to pass in weights as parameters ( $\theta$ )

```

1 from qiskit.circuit.library import ZZFeatureMap, TwoLocal
2
3 data=[0.1 ,0.2]
4
5 zz_feature_map_reference=ZZFeatureMap(feature_dimension=2,reps=2)
6 zz_feature_map_reference= zz_feature_map_reference.assign_parameters(
    data)
7
8 variation_form=TwoLocal(2,[ " ry ", " rz " ], " cz ", reps=2)
9 vqc_ansatz = zz_feature_map_reference.compose(vairation_form)
10 vqc_ansatz.decompose().draw("mpl")

```

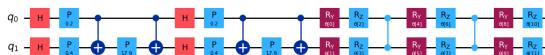


Figure 18.17: Circuit

The high-level variational workload looks as follows:

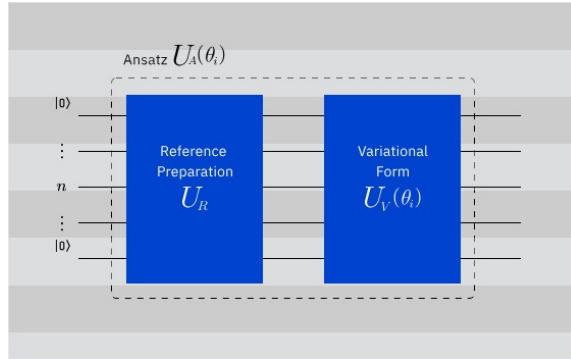


Figure 18.18: Variational workload

For each variation parameter  $\vec{\theta}$ , a different quantum state will be produced. To find the optimal parameters, we need to define a problem-specific cost function to iteratively update our ansatz's parameters.

## 18.4 Cost Functions

We now see how to evaluate a cost function:

1. First, we will learn about Qiskit Runtime primitives
2. Define a cost function  $C(\vec{\theta})$ . This is a problem-specific function that defines the problems goal for the optimizer to minimize (or maximise)
3. Defining a measurement strategy with Qiskit Runtime primitives to optimize speed vs accuracy

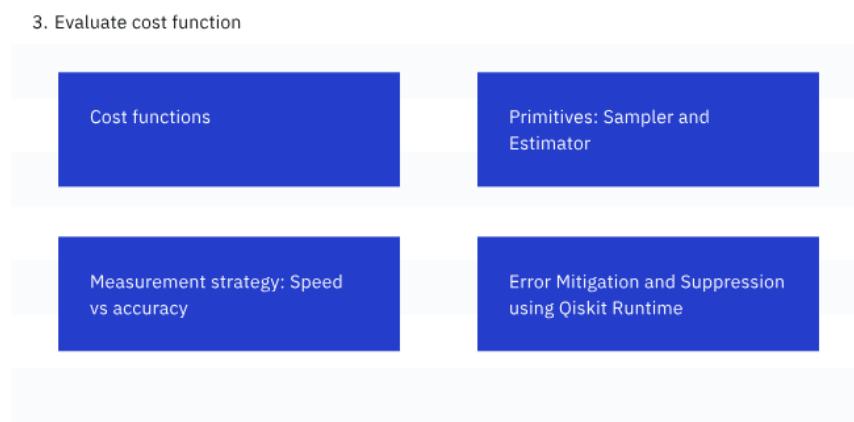


Figure 18.19: Cost function

### 18.4.1 Primitive

All physical systems, whether classical or quantum, exist in different states. For example, a car on a road can have a certain mass, position, speed, or accelerate that characterize its state. Similarly, quantum systems can also have different configurations or states, but they differ from classical system in how we deal with measurements and state evolution. This leads to unique properties such as superposition and entanglement that are exclusive to quantum mechanics. Just like we can describe a car's state using physical properties like speed or acceleration, we can also describe the state of a quantum system using observables, which are mathematical objects.

In quantum mechanics, states are represented by normalized complex column vectors, or kets ( $|\psi\rangle$ ), and **observable are hermitian operators**  $\hat{H} = \hat{H}^\dagger$  that act on the kets. An eigenvector ( $|\lambda\rangle$ ) of an observable is known as an eigenstate. Measuring an observable for one of its eigen states ( $|\lambda\rangle$ ) will give us the corresponding eigenvalue ( $\lambda$ ) as readout.

how to measure a quantum system and what can you measure, Qiskit offers two primitives that can help:

- **Sampler:** Given a quantum state  $|\psi\rangle$ , this primitive obtains the probability of each possible computational basis state.
- **Estimator:** Given a quantum observable  $\hat{H}$  and a state  $|\psi\rangle$ , this primitive computes the expected value of  $\hat{H}$ .

### The Sampler Primitive

The **Sampler** primitive calculates the probability of obtaining each possible state  $|k\rangle$  from the computational basis, given a quantum circuit that prepares the state  $|\psi\rangle$ . It calculates

$$p_k = |\langle k|\psi\rangle|^2 \quad \forall k \in \mathbb{Z}_2^n \equiv \{0, 1, \dots, 2^n - 1\}$$

Qiskit Runtime's Sampler runs the circuit multiple times on a quantum device, performing measurements on each run, and reconstructing the probability distribution from the recovered bit strings. The more runs (or shots) it performs, the more accurate the results will be, but this requires more time and quantum resources.

However, since the number of possible outputs grow exponentially with the number of qubits  $n$  (i.e.  $2^n$ ), the number of shots will need to grow exponentially as well in order to capture a dense probability distribution. Therefore, **Sampler** is only efficient to sparse probability distributions; where the target state  $|\psi\rangle$  must be expressible as a linear combination of the computational basis states, with the number of terms growing at most polynomially with the number of qubits:

$$|\psi\rangle = \sum_k^{Poly(n)} w_k |k\rangle$$

The **Sampler** can also be configured to retrieve probabilities from a subsection of the circuit representing a subset of the total possible states.

### The Estimator Primitive

The **Estimator** primitive calculates the expectation value of an observable  $\hat{H}$  for a quantum state  $|\psi\rangle$ ; where the observable probabilities can be expressed as  $p_\lambda = |\langle \lambda|\psi\rangle|^2$ , being  $|\lambda\rangle$  the eigen states of the observable  $\hat{H}$ . The expectation value is then defined as the average of all possible outcomes  $\lambda$  (i.e. the eigenvalues of the observable) of a measurement of the state  $|\psi\rangle$ , weighted by the corresponding probabilities:

$$\langle\psi|\hat{H}|\psi\rangle = \langle\hat{H}\rangle_\psi := \sum_\lambda \lambda \langle\psi|\lambda\rangle \langle\lambda|\psi\rangle = \sum_\lambda \lambda |\langle\lambda|\psi\rangle|^2 = \sum_\lambda p_\lambda \lambda$$

However, calculating the expectation value of an observable is not always possible, as we often don't know its eigenbasis. Qiskit Runtime's **Estimator** uses a complex algebraic process to estimate the expectation value on a real quantum device by breaking down the observable into a combination of other observables whose eigenbasis we do know.

In simpler terms, **Estimator** breaks down any observable that it doesn't know how to measure into simpler, measurable outcomes called Pauli operators.

Any operator can be expressed as a combination of  $4^n$  Pauli operators.

$$\hat{P}_k := \sigma_{k_{n-1}} \otimes \dots \otimes \sigma_{k_0} \quad \forall k \in \mathbb{Z}_4^n \equiv \{0, 1, \dots, 4^n - 1\}$$

such that

$$\hat{H} = \sum_{k=0}^{4^n-1} w_k \hat{P}_k$$

where  $n$  is the number of qubits  $k \equiv k_{n-1} \dots k_0$  for  $k_l \in \mathbb{Z}_4 \equiv \{0, 1, 2, 3\}$  (i.e. integers base 4), and  $(\sigma_0, \sigma_1, \sigma_2, \sigma_3) := (I, X, Y, Z)$ .

After performing this decomposition, **Estimator** derives a new circuit  $V_k |\psi\rangle$  for each observable  $\hat{P}_k$  (i.e. from the original circuit), to effectively diagonalize the Pauli observable in the computational basis, and calculates the probability  $p_{kj}$  of obtaining each possible output  $j$ . It then looks for the eigenvalue  $\lambda_{kj}$  of  $P_k$  corresponding to each output  $j$ , multiplies by  $w_k$ , and adds all the results together to obtain the expected value of the observable  $\hat{H}$  for the given state  $|\psi\rangle$ .

$$\langle \hat{H} \rangle_\psi = \sum_{k=0}^{4^n-1} w_k \sum_{j=0}^{2^n-1} p_{kj} \lambda_{kj},$$

Since calculating the expectation value of  $4^n$  Paulis is impractical (i.e. exponentially growing), **Estimator** can only be efficient when a large amount of  $w_k$  are zero (i.e. sparse Pauli decomposition instead of dense). Formally we say that, for this computation to be efficiently solvable, the number of non-zero terms has to grow at most polynomially with the number of qubits  $n$ :  $\hat{H} = \sum_k^{\text{Poly}(n)} w_k \hat{P}_k$ . The reader may notice the implicit assumption that probability sampling also needs to be efficient as explained for Sampler, which means

$$\langle H \rangle_\psi = \sum_k^{\text{Poly}(n)} w_k \sum_j^{\text{Poly}(n)} p_{kj} \lambda_{kj}$$

### Guided example to calculate expectation values

Let's assume the single-qubit state  $|+\rangle := H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ , and observable

$$\hat{H} = \begin{pmatrix} -1 & 2 \\ 2 & 1 \end{pmatrix} = 2X - Z$$

with the following theoretical expectation value  $\langle \hat{H} \rangle_+ = \langle +|\hat{H}|+ \rangle = 2$  since we do not know how to measure this observable, we cannot compute its expectation value directly, and we need to re-express it as  $\langle H \rangle_+ = 2\langle X \rangle_+ - \langle Z \rangle_+$ . Which can be shown to evaluate to the same result by virtue of noting that  $\langle +|X|+ \rangle = 1$ , and  $\langle +|Z|+ \rangle = 0$ .

let's see how to compute  $\langle X \rangle_+$  and  $\langle Z \rangle_+$  directly. Since X and Z do not commute (i.e. don't share the same eigen basis), they cannot be measured simultaneously, therefore we need the auxiliary circuits:

```

1 from qiskit import QuantumCircuit
2 from qiskit.quantum_info import SparsePauliOp
3
4 # The following code will work for any other initial single-qubit state
5 original_circuit = QuantumCircuit(1)
6 original_circuit.h(0)
7
8 H = SparsePauliOp([ "X" , "Z" ] , [ 2 , -1 ])
9
10 aux_circuits = []
11 for pauli in H.paulis:
12     aux_circ = original_circuit.copy()
13     aux_circ.barrier()
14     if str(pauli) == "X":
15         aux_circ.h(0)
16     elif str(pauli) == "Y":
17         aux_circ.sdg(0)
18         aux_circ.h(0)
19     else:
20         aux_circ.id(0)
21     aux_circ.measure_all()
22     aux_circuits.append(aux_circ)
23
24 original_circuit.draw("mpl")

```

# Appendix A

## Linear Algebra and Calculus Prerequisites

The following content assumes a high school level understanding of Linear Algebra, Complex numbers, Probability, single-variable Calculus (Multi-Variable Calculus will be a plus) and Differential Equations. It contains all the pre-requisites required for understanding the contents of the given Book. The following references [3], [1], [4] and [2] have been extensively used for the preparation for this appendix. Note that this content assumes that the reader has only high school level exposure of the Quantum Mechanics (De broglie Hypothesis, Schrodinger Equation, Young's Double Slit Experiment, Heisenberg's Uncertainty Principle) but still the important concepts of Quantum Mechanics are explained upto the required depth whenever it is found to be necessary for the explanation of Quantum computing concepts. A college level exposure of Quantum Mechanics will be a plus.

For a quick revision of the concepts of Linear Algebra the reader is suggested to watch youtube lectures by 3 Blue 1 Brown. If it still feels difficult to follow the material, lectures by MIT Prof. Gilbert Strang Lectures on Linear Algebra are highly recommended. For basic Quantum Mechanics concepts, the reader is suggested to watch the lectures by MIT Prof. Allan Adams and for the mathematics of Quantum Mechanics, the lectures by MIT Prof. Barton Zwiebach are highly recommended. For the mathematics of Quantum Computing the short youtube lectures by Quantum Sense and Advanced Maths will be helpful. Hope that this notes help in understanding the basics of Quantum Computing. For the coding part, you can also follow lectures by Qiskit and John Watrous. Many more such youtube lectures are available, lectures by John Preskill where he speaks in a monotonous voice along with his notes on Quantum Computing. Short youtube lectures covering the maths

by David Deutsch and Michael Nielsen. Some good high level view introductory lectures by Scott Aaronson and some amazing videos by Seth Lloyd on the importance of Quantum Computers. Lectures by CERN and many more lecture series on Advanced Quantum Computing and on Quantum Simulations by APCTP.

## A.1 Linear Algebra

### A.1.1 Mathematical Notation

For a complex number  $c = a + \imath b \in \mathbb{C}$  with  $a, b \in \mathbb{R}$  and  $\imath = \sqrt{-1}$ , we use  $c^* = a - \imath b$  (or  $\bar{c}$ ) to denote its complex conjugate. We will also need mathematical notation that is used throughout quantum information. Here we write vectors in a specially way known as the "bra-ket" notation. While it may look a little cumbersome at first sight, it turns out to provide a convenient way of dealing with the many operations we will perform with such vectors. Let's start with two examples,. We write  $|v\rangle \in \mathbb{C}^2$  to denote a vector in a 2-dimensional vector space. For example,

$$|v\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

The "bra" of this vector is the conjugate transpose, which for our example looks like

$$\langle v| = ((|v\rangle)^*)^T = \begin{pmatrix} 1^* \\ 0^* \end{pmatrix}^T = (1 \ 0)$$

The genera definition of the "bra-ket" notation is as follows:

**Definition A.1.1. Ket and Bra notation.** A ket, denoted  $|.\rangle$ , represents a d-dimensional column vector in the complex vector space  $\mathbb{C}^d$ . A bra, denoted  $\langle .|$ , is a d-dimensional row vector equal to the complex conjugate of the corresponding ket, namely

$$\langle .| = (|.\rangle^*)^T$$

where  $*$  denotes the entry-wise conjugate, and  $T$  denotes the transpose.

### A.1.2 Basics of Linear Algebra

**Definition A.1.2.** Linear Transformation: A function  $T : V \rightarrow W$  is called a linear transformation if for all  $|v\rangle, |w\rangle \in V$  and  $a \in \mathbb{F}$  the following properties hold:

1.  $T(|v\rangle + |w\rangle) = T(|v\rangle) + T(|w\rangle)$
2.  $T(a|v\rangle) = aT(|v\rangle)$

**Example A.1.1.** Let  $V$  be a vector space over a field  $\mathbb{F}$  and let  $A$  be a matrix in  $\mathbb{F}^{m \times n}$ . The function  $T : \mathbb{F}^n \rightarrow \mathbb{F}^m$  defined by  $T(|v\rangle) = A|v\rangle$  is a linear transformation.

**Example A.1.2.** Differentiation, Integration, Laplace transform are examples of linear transformations.

**Definition A.1.3.** Composition Transformation: Let  $T : V \rightarrow W$  and  $U : W \rightarrow X$  be linear transformations. The composition transformation  $UT : V \rightarrow X$  is defined by  $(UT)(|v\rangle) = U(T(|v\rangle))$ . If the composition transformation is also a linear transformation then it is called a composite linear transformation. Here, if  $T$  and  $U$  are linear transformations then  $UT$  is also a linear transformation.

**Definition A.1.4.** Inverse Transformation: Let  $T : V \rightarrow W$  be a linear transformation. A linear transformation  $T^{-1} : W \rightarrow V$  is called the inverse transformation of  $T$  if  $TT^{-1} = T^{-1}T = I$  where  $I$  is the identity transformation.

**Definition A.1.5.** Field ( $\mathbb{F}$ ): A field is a set of elements that is closed under two operations, addition and multiplication, and satisfies the following axioms: Let  $a, b, c \in \mathbb{F}$

1. Commutative Property of Addition:  $a + b = b + a$
2. Associative Property of Addition:  $(a + b) + c = a + (b + c)$
3. Distributive Property of Addition:  $a(b + c) = ab + ac$
4. Additive Identity: There exists an element  $0$  in the field such that  $a + 0 = a$
5. Additive Inverse: For every element  $a$  in the field, there exists an element  $-a$  such that  $a + (-a) = 0$
6. Commutative Property of Multiplication:  $ab = ba$
7. Associative Property of Multiplication:  $(ab)c = a(bc)$
8. Distributive Property of Multiplication:  $(a + b)c = ac + bc$
9. Multiplicative Identity: There exists an element  $1$  in the field such that  $a1 = a$

10. Multiplicative Inverse: For every element  $a$  in the field, there exists an element  $a^{-1}$  such that  $aa^{-1} = 1$

Note that it is upto us how we define the addition and multiplication operations on the field  $\mathbb{F}$  but for the purpose of this notes we will assume that the operations are the usual addition and multiplication operations unless stated otherwise. A more formal definition of Field can be defined using group theory but the above definition is sufficient for the purpose of this notes. The complications and more precise definition is left to Mathematicians which can be found in any standard textbook on Abstract Algebra.

**Example A.1.3.** Set of Rational Numbers ( $\mathbb{Q}$ ), Set of Real Numbers ( $\mathbb{R}$ ), Set of Complex Numbers ( $\mathbb{C}$ ) are examples of fields.

**Example A.1.4.** Set of Integers ( $\mathbb{Z}$ ) is not a field as it does not satisfy the multiplicative inverse axiom. Set of Natural Numbers and Whole numbers is not a filed because it does not satisfy the additive inverse axiom.

Generally, we have represented vectors as:  $\vec{v}$ , or with bold letter  $\mathbf{v}$ . but for the purpose of Quantum Mechancis and Quantum Computing we prefer dirac notation which represents vectors as  $|v\rangle$  solely for the purpose of simplifying the notation. The dirac notation is also called as bra-ket notation. The symbol  $|v\rangle$  represents a column vector  $v$  called as ket- $v$  and the symbol  $\langle v|$  represents a row vector called as bra- $v$ .

**Definition A.1.6.** Vector Space: A vector space over a field  $\mathbb{F}$  is a set  $V$  of elements called vectors, that is closed under two operations, addition and scalar multiplication:

1. Vector Addition: For every pair of vectors  $|u\rangle, |v\rangle \in V$ , there exists a vector  $|u\rangle + |v\rangle \in V$  called the sum of  $|u\rangle$  and  $|v\rangle$
2. Scalar Multiplication: For every vector  $|u\rangle \in V$  and every scalar  $a \in \mathbb{F}$ , there exists a vector  $a|u\rangle \in V$  called the scalar multiple of  $|u\rangle$  by  $a$

and satisfies the following axioms: Let  $|u\rangle, |v\rangle, |w\rangle \in V$  and  $a, b \in \mathbb{F}$

1. Closure Property over Vector Addition:  $|u\rangle + |v\rangle \in V$
2. Commutative Property of Addition:  $|u\rangle + |v\rangle = |v\rangle + |u\rangle$
3. Associative Property of Addition:  $(|u\rangle + |v\rangle) + |w\rangle = |u\rangle + (|v\rangle + |w\rangle)$
4. Additive Identity: There exists a vector  $|0\rangle$  in  $V$  such that  $|u\rangle + |0\rangle = |u\rangle$

5. Additive Inverse: For every vector  $|u\rangle$  in  $V$ , there exists a vector  $-|u\rangle$  in  $V$  such that  $|u\rangle + (-|u\rangle) = |0\rangle$
6. Closure Property over Scalar Multiplication:  $a|u\rangle \in V$
7. Distributive Property of Scalar Multiplication over Vector Addition:  $a(|u\rangle + |v\rangle) = a|u\rangle + a|v\rangle$
8. Distributive Property of Scalar Multiplication over Field Addition:  $(a+b)|u\rangle = a|u\rangle + b|u\rangle$
9. Associative Property of Scalar Multiplication:  $a(b|u\rangle) = (ab)|u\rangle$
10. Multiplicative Identity: There exists a scalar  $1$  in  $\mathbb{F}$  such that  $1|u\rangle = |u\rangle$

The above axioms imply

1.  $|0\rangle$  is unique i.e. if  $|0'\rangle$  has all the properties of  $|0\rangle$  then  $|0'\rangle = |0\rangle$
2.  $|-u\rangle = -|u\rangle$
3.  $|-u\rangle$  is the unique additive inverse of  $|u\rangle$

**Example A.1.5.** The set of Real numbers  $\mathbb{R}$  is a vector space over the field of Real numbers  $\mathbb{R}$ . It is called Real Vector Space. The set of Complex numbers  $\mathbb{C}$  is a vector space over the field of Complex numbers  $\mathbb{C}$ . It is called Complex Vector Space.

Note the adjective Real and Complex before a vector space is used to denote the field over which the vector space is defined. In this notes we will be mostly dealing with the Complex Vector Space since the Real Vector Space is a special case of the Complex Vector Space.

Any element that satisfies the above said axioms (A.1.6) of a vector space is called a vector. When we generally refer to a vector, picture of arrow comes to our mind but arrows are just one of the manifestation of a representation of a Real Vector Space. Thus to make the reader digress from thinking of vectors as arrows, and be more general, below given are few examples of manifestations of vectors in a vector space.

**Example A.1.6.** Matrices are vectors in the vector space of  $n \times n$  matrices over a field  $\mathbb{F}$ . The vector addition is defined as the element-wise addition of the matrices and the scalar multiplication is defined as the element-wise multiplication of the matrix by the scalar. It satisfies the axioms of vector space (refer definition A.1.6). Thus, they are elements of vector space. Note that matrices have no notion of length (magnitude) and direction unlike arrows.

**Example A.1.7.** Functions are vectors in the vector space of functions over a field  $\mathbb{F}$ . The vector addition and scalar multiplication is defined as the usual addition of two functions and scalar multiplication is simply multiplying the function by the scalar respectively. It satisfies the above said axioms of vector space. Thus, they are elements of vector space. Note that functions have no notion of length (magnitude) and direction unlike arrows. The functions are an example of infinite dimensional vector space unlike discrete dimensional vector space (such as matrices, vectors).

Thus when we say vector we refer to any element of a vector space and not just the arrows. The arrows are just one of the manifestations of vectors in a vector space and the other examples include matrices, functions. In Quantum mechanics we generally deal with Hilbert Space (Complex Vector Space with inner product defined, more about it later). In Quantum Computing, we deal with finite dimensional complex vector spaces. Thus, it could not be imagined as arrows since they deal with real vector spaces (but imagining as arrows helps in general).

**Definition A.1.7. Subspace:** A subset  $W$  of a vector space  $V$  over a field  $\mathbb{F}$  is called a subspace of  $V$  if  $W$  is itself a vector space over  $\mathbb{F}$  with the same operations of addition and scalar multiplication defined on  $V$ . Note that a subspace must always contain zero vector  $|0\rangle$  of the vector space  $V$  to satisfy the additive identity axiom.

**Example A.1.8.** Set of all real symmetric matrices is a subspace of the vector space of all real matrices.

**Definition A.1.8. Linear Combination:** Let  $V$  be a vector space over a field  $\mathbb{F}$  and let  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  be vectors in  $V$ . A vector of the form

$$a_1|v_1\rangle + a_2|v_2\rangle + \dots + a_n|v_n\rangle \quad (\text{A.1})$$

where  $a_1, a_2, \dots, a_n \in \mathbb{F}$  is called a linear combination of  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$

**Example A.1.9.**  $3x^2 + 2x + 1 = 0$  is a linear combination of  $|v_1\rangle = x^2$ ,  $|v_2\rangle = x$  and  $|v_3\rangle = 1$  in the vector space of polynomials with coefficients in  $\mathbb{R}$  as the field.

$$3x^2 + 2x + 1 = 3|v_1\rangle + 2|v_2\rangle + 1|v_3\rangle$$

**Example A.1.10.** Consider the matrix  $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$  in the vector space of  $2 \times 2$  matrices over the field  $\mathbb{R}$ . It can be written as a linear combination of the matrices  $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ ,  $\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ ,  $\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$  and  $\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$  as

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

**Definition A.1.9. Span:** Let  $V$  be a vector space over a field  $\mathbb{F}$  and let  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  be vectors in  $V$ . The set of all linear combinations of  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  is called the span of  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  and is denoted by  $Span(|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle)$

**Definition A.1.10. Linear Independence:** Let  $V$  be a vector space over a field  $\mathbb{F}$  and let  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  be vectors in  $V$ . The vectors  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  are said to be linearly independent if the only solution to the equation

$$a_1|v_1\rangle + a_2|v_2\rangle + \dots + a_n|v_n\rangle = |0\rangle \quad (\text{A.2})$$

is trivial solution  $a_1 = a_2 = \dots = a_n = 0$

If the vector equation has non-trivial solution (e.e infinitely many solutions or at least one of the  $a_i \neq 0$ ) then the vectors are said to be linearly dependent.

**Corollary A.1.1.** *A set  $S$  with two or more vectors is not linearly independent i.e. linearly dependent iff at least one vector is expressible as a linear combination of other vectors in  $S$*

**Corollary A.1.2.** *If finite set  $S$  contains zero vector  $|0\rangle$  then the set  $S$  is linearly dependent.*

**Definition A.1.11.** Basis: Let  $V$  be a vector space over a field  $\mathbb{F}$ . A set of vectors  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  in  $V$  is called a basis of  $V$  if

1.  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  are linearly independent
2.  $Span(|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle) = V$

**Example A.1.11.** A collection of  $n$  linearly independent vector alswas spans  $\mathbb{R}^n$  and is called a basis of  $\mathbb{R}^n$ .

**Definition A.1.12.** Dimension: Let  $V$  be a vector space over a field  $\mathbb{F}$ . The number of vectors in a basis of  $V$  is called the dimension of  $V$  and is denoted by  $\dim(V)$  Note that this is because the number of vectors in a basis of any Vector space is unique.

**Theorem A.1.3.** *The coefficients of expansion  $v_i$  of vector  $|v\rangle$  in terms of linearly independent basis  $|i\rangle$  are called components of the vector in that basis. It is always a unique expansion in a given basis since the basis is linearly indepnde and only a trivial linear relation between them can exist. Given a basis components are unique, but if we change the basis, the components will change.*

Note that  $|v\rangle$  is an abstract which satisfies various relations. When we choose a basis the vectors assume a concrete form and the relation between vectors is satisfied by the components. On changing basis, the components will change in numerical value, but the relation between them expressing the equality still holds between the set of new components.

Addition of two vectors in a basis (both the vectors are expressed in the same basis):

$$\begin{aligned} |v\rangle &= \sum_{i=1}^n v_i |i\rangle, |w\rangle = \sum_{i=1}^n w_i |i\rangle \\ |v\rangle + |w\rangle &= \sum_{i=1}^n (v_i + w_i) |i\rangle \end{aligned}$$

To add two vectors, add their corresponding components in the same basis.

Scalar multiplication of a vector in a basis:

$$a |v\rangle = a \sum_{i=1}^n v_i |i\rangle = \sum_{i=1}^n (av_i) |i\rangle$$

To multiply a vector by a scalar, multiply each component by the scalar.

### A.1.3 Inner Product

Similar to the definition of dot product defined in  $\mathbb{R}^n$  we define inner product in a vector space. The inner product is a generalization of the dot product in  $\mathbb{C}^n$ . The inner product is a function that takes two vectors as input and returns a scalar as output. The inner product is denoted by  $\langle u|v\rangle$  and is defined as follows:

**Definition A.1.13.** Inner Product: Let  $V$  be a vector space over a field  $\mathbb{F}$ . An inner product on  $V$  is a function that assigns to each pair of vectors  $|u\rangle, |v\rangle \in V$  a scalar  $\langle u|v\rangle \in \mathbb{F}$  such that for all  $|u\rangle, |v\rangle, |w\rangle \in V$  and  $a, b \in \mathbb{F}$  the following properties hold:

1. Conjugate Symmetry:  $\langle u|v\rangle = \langle v|u\rangle^*$
2. Linearity in the second argument:  $\langle v|au + bw\rangle = a\langle v|u\rangle + b\langle v|w\rangle$
3. Positive Definite:  $\langle v|v\rangle \geq 0$  and  $\langle v|v\rangle = 0$  iff  $|v\rangle = |0\rangle$

**A vector space with an inner product defined is called an inner product space.**

**Example A.1.12.** In  $\mathbb{R}^n$  as Field the inner product is defined as  $\langle u|v \rangle = \sum_{i=1}^n u_i v_i$  and holds the following properties:

1. Symmetry:  $\langle u|v \rangle = \langle v|u \rangle^*$
2. Linearity:  $\langle au + bw|v \rangle = a \langle u|v \rangle + b \langle w|v \rangle = \langle v|au + bw \rangle$
3. Positive Definite:  $\langle v|v \rangle = \sum_{i=1}^n v_i^2 \geq 0$  and  $\langle v|v \rangle = 0$  iff  $|v\rangle = |0\rangle$

Here the properties of inner product are reduced since it is defined on Real numbers. The conjugate symmetry Property is reduced to Symmetry Property. The Linearity in first argument reduces to Linearity in both arguments. The Positive Definite property remains the same.

**Antilinearity:** The inner product is linear in the second argument and antilinear in the first argument. The antilinearity in the first argument is defined as follows:

$$\begin{aligned}\langle av + bw|u \rangle &= \langle u|av + bw \rangle^* \quad (\text{Using Property 1: Conjugate Symmetry}) \\ &= (a \langle v|u \rangle + b \langle w|u \rangle)^* \quad (\text{using Property 2: Linearity in second argument}) \\ &= a^* \langle v|u \rangle + b^* \langle w|u \rangle \quad (\text{Using the properties of complex conjugate})\end{aligned}$$

where  $a^*, b^*$  is the complex conjugate of  $a$  and  $b$  respectively.

Inner products of a linear superposition with another vector is the corresponding superposition of inner products if the superposition occurs in second factor, while it is the superposition with all coefficients conjugated if the superposition occurs in the first factor.

**Another way in which inner product can be thought of is as projections.** The inner product  $\langle u|v \rangle$  is the projection of  $|v\rangle$  onto  $|u\rangle$ . Similarly, we can say that the inner product  $\langle v|u \rangle$  is the projection of  $|u\rangle$  onto  $|v\rangle$ . Note that unlike in the case of real vector spaces here the projections of  $a$  on  $b$  is not the same as the projection of  $b$  on  $a$ . here, the projections are complex conjugates of each other as can be clearly observed since the inner product is conjugate symmetric ( $\langle u|v \rangle = \langle v|u \rangle^*$ ). Thus, the projection of  $u$  on  $v$  is a complex conjugate of the projection of  $v$  on  $u$ .

**Definition A.1.14. Absolute value of a complex number.** Consider a complex number  $z \in \mathbb{C}$  expressed as  $z = x + iy$  where  $x, y \in \mathbb{R}$  are real numbers representing

the real and imaginary parts of  $z$  respectively. The absolute value, or otherwise known as modulus of  $z$  is given by

$$|z| = \sqrt{z^* z} = \sqrt{x^2 + y^2}$$

**Example A.1.13.** For example, for  $z = 1, 2$  its absolute value is given by  $|z| = \sqrt{1^2 + 2^2} = \sqrt{5}$ .

**Definition A.1.15.** Norm (length) of a vector: Let  $V$  be a vector space over a field  $\mathbb{F}$ . The norm of a vector  $|v\rangle \in V$  is defined as  $\| |v\rangle \| = \sqrt{\langle v|v \rangle}$ . A vector which has a norm of 1 is called a unit vector.

**Example A.1.14.** Consider a ket  $|v\rangle = \frac{1}{2} \begin{pmatrix} 1 + \iota \\ 1 - \iota \end{pmatrix} \in \mathbb{C}^2$ . The corresponding bra is given by  $\langle v| = \frac{1}{2} (1 - \iota \quad 1 + \iota)$ , and the length of  $|v\rangle$  is

$$\sqrt{\langle v|v \rangle} = \sqrt{\frac{1}{4} 2(1 + \iota)(1 - \iota)} = \sqrt{\frac{1}{2}(1 + \iota - \iota - \iota^2)} = \sqrt{\frac{1}{2} 2} = 1$$

**Definition A.1.16.** Orthogonality: Let  $V$  be a vector space over a field  $\mathbb{F}$ . Two vectors  $|u\rangle, |v\rangle \in V$  are said to be orthogonal if  $\langle u|v \rangle = 0$

**Definition A.1.17.** Orthonormal: Let  $V$  be a vector space over a field  $\mathbb{F}$ . Two vectors  $|u\rangle, |v\rangle \in V$  are said to be orthonormal if they are orthogonal and have unit norm.

**Definition A.1.18.** Orthonormal Basis: Let  $V$  be a vector space over a field  $\mathbb{F}$ . A basis  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  of  $V$  is called an orthonormal basis if

1.  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  are orthogonal
2.  $\| |v_i\rangle \| = \sqrt{\langle v|v \rangle} = 1$  for all  $i = 1, 2, \dots, n$

**Example A.1.15.** A general formula of Inner Product: Let  $|u\rangle = \sum_{i=1}^n u_i |i\rangle$  and  $|v\rangle = \sum_{j=1}^n v_j |j\rangle$  be two vectors in a vector space  $V$  over a field  $\mathbb{F}$ . Then the inner product of  $|u\rangle$  and  $|v\rangle$  is given by

$$\langle u|v \rangle = \sum_{j=1}^n \sum_{i=1}^n u_i^* v_j \langle i|j \rangle$$

If the basis is orthonormal then the  $\langle i|j \rangle$  evaluates as follows:

$$\langle i|j \rangle = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

Thus, in the orthonormal basis this gets simplified to

$$\langle u|v \rangle = \sum_{i=1}^n u_i^* v_i$$

Thus, defined inner product satisfies the above axioms of inner product. For norm of a vector  $|v\rangle$  in an orthonormal basis is given by

$$\begin{aligned}\| |v\rangle \| &= \sqrt{\langle v|v \rangle} = \sqrt{\sum_{i=1}^n v_i^* v_i} = \sqrt{\sum_{i=1}^n |v_i|^2} \\ \langle v|v \rangle &= \sum_{i=1}^n |v_i|^2 \geq 0\end{aligned}$$

thus, it is used to define the length of the vector. Also note that  $|v_i|$  is the modulus of a complex number (thus  $|v_i| \geq 0$ ).

Now note that the inner product can be written as matrix multiplication between

two vectors. Let  $|u\rangle = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}$  and  $|v\rangle = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$  be two vectors in a vector space  $V$

over a field  $\mathbb{C}$  with basis vectors as  $|i\rangle = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$  where 1 is in the  $i$ th row position.

Then the inner product of  $|u\rangle$  and  $|v\rangle$  is given by

$$\langle u|v \rangle = (u_1^* \ u_2^* \ \dots \ u_n^*) \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} = (u_1^* \ u_2^* \ \dots \ u_n^*) \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} = \sum_{i=1}^n u_i^* v_i$$

Thus, the inner product can be written as matrix multiplication between two vectors. Note that here  $\langle u|$  is called a bra vector and we can think of it as lying in

some other vector space defined by the basis vectors:  $\langle i | = (0 \ \dots \ 0 \ 1 \ 0 \ \dots \ 0)$  where 1 is in the  $i$ th column position. Thus we have a vector space of  $|v\rangle$  and another vector space of  $\langle v|$ , these two spaces are called dual spaces. We can convert a vector from one space to other by performing the adjoint operation (transpose conjugate operation). Thus,  $\langle v| = ((|v\rangle)^T)^*$  or  $|v\rangle = ((\langle v|)^T)^*$ .

Properties of Adjoint Operation:

1.  $(a|v\rangle + b|w\rangle)^\dagger = \langle v|a^* + \langle w|b^*$
2.  $(|v\rangle^\dagger)^\dagger = |v\rangle$

**Example A.1.16.** note that the inner product of two vectors  $|v_1\rangle, |v_2\rangle \in \mathbb{C}^d$  is in general a complex number. Later on, we shall see that the modulus squared of the inner product  $|\langle v_1|v_2\rangle|^2$  is of much significance. As an example, let us consider the inner product of the vectors  $|v\rangle$  and  $|w\rangle$  given as

$$|v\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \text{ket } w = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

We have

$$\langle v|w\rangle = (1 \ 0) \begin{pmatrix} 2 \\ 3 \end{pmatrix} = 1 \cdot 2 + 0 \cdot 3 = 2$$

Note that  $|\langle v_1|v_2\rangle|^2 = \langle v_1|v_2\rangle \langle v_2|v_1\rangle$ .

#### A.1.4 Gram-Schmidt Orthogonalization

Given a set of linearly independent vectors  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  in a vector space  $V$  over a field  $\mathbb{F}$ , the Gram-Schmidt orthogonalization process constructs an orthonormal basis  $|u_1\rangle, |u_2\rangle, \dots, |u_n\rangle$  of  $V$  from the given set of linearly independent vectors. The process is as follows:

1. Let  $|u_1\rangle = \frac{|v_1\rangle}{\|v_1\|}$
2. For  $i = 2, 3, \dots, n$  do
  - (a) Let  $|u_i\rangle = |v_i\rangle - \sum_{j=1}^{i-1} \langle u_j|v_i\rangle |u_j\rangle$
  - (b) Normalize the vector  $|u_i\rangle$ :  $|u_i\rangle = \frac{|u_i\rangle}{\|u_i\|}$

*Proof.* Proof that Gram-Schmidt Orthogonalization process constructs an orthonormal basis of  $V$  from a given set of linearly independent vectors  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  in a vector space  $V$  over a field  $\mathbb{F}$ . The proof can be shown by using induction.

**Base Case**

For  $i = 1$ ,  $|u_1\rangle = \frac{|v_1\rangle}{\|v_1\|}$  is a unit vector and thus is an orthonormal basis.

**Inductive Hypothesis**

Assume that for some  $k \geq 1$ , the vectors  $|u_1\rangle, |u_2\rangle, \dots, |u_k\rangle$  are orthonormal.

**Inductive Step**

We need to show that the vector  $|u_{k+1}\rangle$  is orthonormal. We have

$$\langle u_i | u_j \rangle = \delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

for  $i, j = 1, 2, \dots, k$ . Thus, we need to show that  $\langle u_i | u_{k+1} \rangle = 0$  for  $i = 1, 2, \dots, k$ .

We have

$$\langle u_i | u_{k+1} \rangle = \langle u_i | v_{k+1} \rangle - \sum_{j=1}^k \langle u_j | v_{k+1} \rangle \langle u_i | u_j \rangle$$

Now,  $\langle u_i | u_j \rangle = \delta_{ij}$  and thus the above equation becomes,

$$= \langle u_i | v_{k+1} \rangle - \langle u_i | v_{k+1} \rangle \langle u_i | u_i \rangle$$

Using the fact that  $|u_i\rangle$  is a unit vector (from inductive hypothesis that  $|u_1\rangle, \dots, |u_k\rangle$  is an orthonormal basis), we have

$$= \langle v_{k+1} | u_i \rangle - \langle u_i | v_{k+1} \rangle$$

$$= 0$$

Thus,  $|u_{k+1}\rangle$  is orthogonal to  $|u_1\rangle, |u_2\rangle, \dots, |u_k\rangle$ . Now, we need to show that  $|u_{k+1}\rangle$  is a unit vector. We have

$$\| |u_{k+1}\rangle \| = \sqrt{\langle u_{k+1} | u_{k+1} \rangle} = \sqrt{\langle v_{k+1} | u_{k+1} \rangle}$$

Thus,  $|u_{k+1}\rangle$  is a unit vector. Thus, by induction, the Gram-Schmidt orthogonalization process constructs an orthonormal basis of  $V$  from a given set of linearly independent vectors  $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$  in a vector space  $V$  over a field  $\mathbb{F}$ .

□

### A.1.5 Cauchy-Schwarz and Traingle Inequalities

**Theorem A.1.4.** *The Schwarz inequality: Dot product of two vectors cannot exceed the product of their lengths.*

$$|\langle V|W \rangle| \leq \sqrt{\langle V|V \rangle \langle W|W \rangle}$$

*Proof.* Note that this is obviously true for arrows. Now say,

$$|Z\rangle = |V\rangle - \frac{\langle W|V\rangle}{|W|^2} |W\rangle$$

$$\langle Z| = \langle V| - \frac{\langle V|W\rangle}{|W|^2} \langle W|$$

Note here we are required to find  $\langle Z|Z \rangle$ . Thus we can either write the bra form and then proceed solving with direct inner product or we can use the ket version in the first factor and then use the antilinearity axiom (for superposition in the first factor) to proceed further. Then, we have

$$\langle Z|Z \rangle = \langle V|V \rangle - \frac{\langle V|W\rangle \langle W|V\rangle}{|W|^2} - \frac{\langle W|V\rangle \langle V|W\rangle}{|W|^2} + \frac{\langle V|W\rangle \langle W|V\rangle \langle W|W\rangle}{|W|^4}$$

Now  $\langle Z|Z \rangle \geq 0$  from axiom. ( $\langle W|W \rangle = |W|^2$ ) Thus,

$$|\langle V|V \rangle| \geq \frac{\langle V|W\rangle \langle W|V\rangle}{|W|^2}$$

$$\langle V|V \rangle \langle W|W \rangle \geq |\langle V|W \rangle|^2$$

$$|\langle V|W \rangle|^2 \leq \langle V|V \rangle \langle W|W \rangle$$

$$|\langle V|W \rangle| \leq \sqrt{\langle V|V \rangle \langle W|W \rangle}$$

Hence proved. □

**Theorem A.1.5.** *The Triangle Inequality: the length of a sum cannot exceed the sum of the lengths.*

$$|V + W| \leq |V| + |W|$$

*Proof.* Note that this is obviously true for arrows. Now say,

$$|Z\rangle = |V\rangle + |W\rangle$$

$$\langle Z| = \langle V| + \langle W|$$

Thus, we have

$$\begin{aligned}\langle Z|Z\rangle &= |Z|^2 = \langle V|V\rangle + \langle W|W\rangle + \langle V|W\rangle + \langle W|V\rangle \\ |V+W|^2 &= |V|^2 + |W|^2 + 2\operatorname{Re}(\langle V|W\rangle)\end{aligned}$$

Now using,  $\operatorname{Re}(\langle V|W\rangle) \leq |\langle V|W\rangle|$ , we get

$$|V+W|^2 \leq |V|^2 + |W|^2 + 2|\langle V|W\rangle|$$

Using Schwarz Inequality, we get

$$\begin{aligned}|V+W|^2 &\leq |V|^2 + |W|^2 + 2|V||W| \\ |V+W|^2 &\leq (|V| + |W|)^2 \\ |V+W| &\leq |V| + |W|\end{aligned}$$

Hence, proved.  $\square$

### A.1.6 Linear Operators and Matrix Representation

**Definition A.1.19.** Linear Operator: Let  $V$  and  $W$  be vector spaces over a field  $\mathbb{F}$ . A function  $A : V \rightarrow W$  is called a linear operator if for all  $|v\rangle, |w\rangle \in V$  and  $a \in \mathbb{F}$  the following properties hold:

1.  $A(|v\rangle + |w\rangle) = A(|v\rangle) + A(|w\rangle)$
2.  $A(a|v\rangle) = aA(|v\rangle)$

In general, when we say a linear operator is defined on a vector space  $V$ , we mean that  $A$  is a linear operator from  $V$  to  $V$ .

Once the action of Linear Operators on the basis vectors is known, their action on any vector space is determined. For basis  $|1\rangle, |2\rangle, \dots, |n\rangle$  of a vector space  $V$  over a field  $\mathbb{F}$ , the action of a linear operator  $A$  on any vector  $|v\rangle = \sum_{i=1}^n v_i |i\rangle$  say transforms the basis vectors to  $|1'\rangle, |2'\rangle, \dots, |n'\rangle$  and thus, the vector  $|v\rangle$  to  $|v'\rangle$  as follows:

$$A(|v\rangle) = A\left(\sum_{i=1}^n v_i |i\rangle\right) = \sum_{i=1}^n v_i A(|i\rangle) = \sum_{i=1}^n v_i |i'\rangle = |v'\rangle$$

Thus, once if we know the action of a linear operator on the basis vectors, we can determine the action of the linear operator on any vector in the vector space. It is

the same linear combination of the vector in the new basis as it was in the old basis. Note we haven't talked anything about writing this operators in matrix format till now and so one must not think in terms of matrix and confuse. The above statement simply says that say given a basis vectors of Vector Space V and a linear operator A, then the action of A on any vector in the vector space is determined by the action of A on the basis vectors. In other words, given the basis vectors  $|1\rangle, |2\rangle, \dots, |n\rangle$  of Vector space V and a linear operator A and its transformed basis vectors  $|1'\rangle, |2'\rangle, \dots, |m'\rangle$  of Vector Space W, we can find what the any vector in the Vector Space V will be in the Vector Space W when we apply the Operator A on the vector. The matrix representation and operators are equivalent for the purpose of this notes and is applicable in Quantum Computing.

**Definition A.1.20.** Matrix Representation of a Linear Operator: Let  $V$  be a vector space over a field  $\mathbb{F}$  with basis  $|1\rangle, |2\rangle, \dots, |n\rangle$  and let  $W$  be a vector space over  $\mathbb{F}$  with basis  $|1'\rangle, |2'\rangle, \dots, |m'\rangle$ . Let  $A$  be a linear operator from  $V$  to  $W$ . The matrix representation of  $A$  with respect to the basis  $|1\rangle, |2\rangle, \dots, |n\rangle$  of  $V$  and the basis  $|1'\rangle, |2'\rangle, \dots, |m'\rangle$  of  $W$  is the  $m \times n$  matrix  $[A]$  whose elements are given by

$$[A]_{ij} = \langle i' | A | j \rangle$$

which is the ith row and jth column entry.

To explain this in more depth. We must know what the operator does to the basis vectors. Then to represent this operator in some matrix representation we must first specify the input and ouput basis for the matrix to map. In general, the input and ouput basis are the same and they are the same standard orthonormal basis that we use (i.e.  $|1\rangle = 1, 0, 0, \dots, 0$ ,  $|2\rangle = 0, 1, 0, \dots, 0$  and so on). Thus, the matrix representation of the operator in this basis will be given as follows:

$$[A] = \begin{pmatrix} \langle 1' | A | 1 \rangle & \langle 1' | A | 2 \rangle & \dots & \langle 1' | A | n \rangle \\ \langle 2' | A | 1 \rangle & \langle 2' | A | 2 \rangle & \dots & \langle 2' | A | n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle m' | A | 1 \rangle & \langle m' | A | 2 \rangle & \dots & \langle m' | A | n \rangle \end{pmatrix}$$

this indicates that the input basis here is chosen as  $|1\rangle, |2\rangle, \dots, |n\rangle$  and the output basis is chosen as  $|1'\rangle, |2'\rangle, \dots, |m'\rangle$ . Note that the jth column represents the components of the jth transformed basis vector  $|j'\rangle = A |j\rangle$ . Thus the value  $\langle i' | A | j \rangle$  is a scalar value which is the component of the jth basis vector  $|j\rangle$  after being transformed by the action of opertor A to  $A |j\rangle$  on the ith transformed basis vector  $|i'\rangle$ .

Generally for the purpose of this notes and for the most of the applications we assume the input and output basis to be the standard orthonormal basis. Thus, the problems we will face will be of the kind that we will be given operations of an operator on some vectors (sufficient enough to find its matrix representation) and then we will be asked to find its matrix representation in some input and output basis. Consider the following example for more clarity.

**Example A.1.17.** Suppose  $V$  is a vector space with basis vectors  $|0\rangle$  and  $|1\rangle$ , and  $A$  is a linear operator from  $V$  to  $V$  such that  $A|0\rangle = |1\rangle$  and  $A|1\rangle = |0\rangle$ . Find the matrix representation of  $A$  in the basis  $|0\rangle$  and  $|1\rangle$  (since it doesn't say input and ouput basis, we assume that the input and ouput basis are the same and they are  $|0\rangle$  and  $|1\rangle$ ). Also find the matrix representation of  $A$  in the basis  $|+\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$  and  $|-\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$  represented in the standard orthonormal basis. Note that the basis  $|+\rangle$  and  $|-\rangle$  are orthonormal.

Solution: The matrix representation of  $A$  in the basis  $|0\rangle$  and  $|1\rangle$  is given by

$$[A] = \begin{pmatrix} \langle 0|A|0\rangle & \langle 0|A|1\rangle \\ \langle 1|A|0\rangle & \langle 1|A|1\rangle \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

The matrix representation of  $A$  in the basis  $|+\rangle$  and  $|-\rangle$  is given by

$$[A] = \begin{pmatrix} \langle +|A|+ \rangle & \langle +|A|- \rangle \\ \langle -|A|+ \rangle & \langle -|A|- \rangle \end{pmatrix}$$

Note that now we are not given the action of operator  $A$  on the vector  $|+\rangle$  and  $|-\rangle$ , thus we must find it first.

$$A|+\rangle = A \left( \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = |+\rangle$$

$$A|-\rangle = A \left( \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \right) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = -\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = -|-\rangle$$

Thus the matrix representation of  $A$  in the basis  $|+\rangle$  and  $|-\rangle$  using  $A|+\rangle = |+\rangle$  and  $A|-\rangle = -|-\rangle$  is given by

$$[A] = \begin{pmatrix} \langle +|A|+ \rangle & \langle +|A|- \rangle \\ \langle -|A|+ \rangle & \langle -|A|- \rangle \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Thus, given action of linear operators on some vectors (assumed sufficient enough to find the matrix representation) we can find its matrix representation in any input and output basis as shown by the above procedure. The matrix may be numerically different in different basis. Note that once the matrix is specified in a given input and output basis it can operate only on the vectors that are specified in the same input basis and the output of the operation will result in a vector specified in the given ouput basis. Note that to make the connection between matrices and linear operators one must specify a set of input and ouput basis states for the input and output vectors spaces of the linear operator. The matrix representation can thus be found for a linear operator using the above mentioned procedure. On a side note, note that it is not at all necessary for the input and output basis to be the same or even orthonormal as that property has not been used anywhere while deriving the formulation, the condition that needs to be satisfied is just that the basis vectors must be specified for the linear operator's input and output vector spaces.

From now on, when we speak of a matrix representation for a linear operator, we mean a matrix representation with respect to orthonormal input and ouput bases. We also use the convention that if the input and output spaces for a linear operator are the same, then the input and ouput bases are the same, unless noted otherwise. Some of the Important linear Operators:

1. Identity Operator: The identity operator  $I$  is defined as  $I|v\rangle = |v\rangle$  for all  $|v\rangle \in V$ . The matrix representation of the identity operator in any basis is the identity matrix.

$$\langle i| I |j\rangle = \langle i|j\rangle = \delta_{ij}$$

where  $\delta_{ij}$  is the Kronecker delta function.

2. Zero Operator: The zero operator  $0$  is defined as  $0|v\rangle = |0\rangle$  for all  $|v\rangle \in V$ . The matrix representation of the zero operator in any basis is the zero matrix.
3. Projection Operator: Consider a vector  $|v\rangle$  which can be written in some basis as:

$$|v\rangle = \sum_{i=1}^n v_i |i\rangle$$

This can be further simplified using the inner product definition as  $v_i$  is the projection of the  $|v\rangle$  onto the  $i$ th basis vector  $|i\rangle$ . Thus,  $v_i = \langle i|v\rangle$ . On substituting

we get,

$$|v\rangle = \sum_{i=1}^n \langle i|v\rangle |i\rangle$$

Rearranging we get,

$$|v\rangle = \sum_{i=1}^n |i\rangle \langle i|v\rangle = \left( \sum_{i=1}^n |i\rangle \langle i| \right) |v\rangle$$

Let  $\mathbb{P}_i = |i\rangle \langle i|$  be the projection operator which projects any vector  $|v\rangle$  onto the subspace spanned by the basis vector  $|i\rangle$ . Thus,  $\mathbb{P}_i |v\rangle = |i\rangle \langle i|v\rangle$ . Thus,  $\mathbb{P} = \sum_{i=1}^n \mathbb{P}_i = \sum_{i=1}^n |i\rangle \langle i| = I$  (Identity Operator) is called the **Completeness Relation**, and  $\mathbb{P}|v\rangle = |v\rangle$ . Thus,  $\mathbb{P}$  is a projection operator which projects any vector  $|v\rangle$  onto the subspace spanned by the basis vectors  $|i\rangle$ . Projection operator acting on bra  $\langle v|$  is  $\langle v| \mathbb{P}_i = \langle i| v_i^*$ . Now,

$$\mathbb{P}_i \mathbb{P}_j = |i\rangle \langle i|j\rangle \langle j| = \delta_{ij} \mathbb{P}_j$$

this equation indicates that once  $\mathbb{P}_i$  projects out the part of  $|V\rangle$  along  $|i\rangle$ , further applications of  $\mathbb{P}_i$  make no difference; and the subsequent applications of  $\mathbb{P}_i (i \neq j)$  will result in zero, since a vector entirely along  $|i\rangle$  cannot have a projection along a perpendicular direction  $|j\rangle$ . Thus,  $P^2 = P$  property must be satisfied by any projection matrix. Note that here we have assumed that the vectors to be projected are on orthonormal basis.

In the standard orthonormal basis the projection operator  $\mathbb{P}_i = |i\rangle \langle i|$  will be 1 at the  $i$ th position in the diagonal and zero every where else.

$$\mathbb{P}_i = |i\rangle \langle i| = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix} \begin{pmatrix} 0 & \dots & 0 & 0 \\ \dots & 0 & 1 & 0 \\ 0 & 1 & 0 & \dots \\ \vdots & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & \dots & 0 & 0 \\ \vdots & 1 & \dots & 0 \\ 0 & \dots & 0 & 0 \end{pmatrix}$$

Thus, upon adding all the individual projection operator on  $i$  we get the identity operator. The matrix element of the projection operators are thus given as:

$$(\mathbb{P}_i)_{kl} = \langle k|i\rangle \langle i|l\rangle = \delta_{ki} \delta_{il} = \delta_{kl}$$

$I - P$  is also a projection operator which projects onto the subspace orthogonal to the subspace spanned by the basis vector  $|i\rangle$  and is called orthogonal complement.

4. Rotation operator:  $R(\theta)$  on  $\mathbb{R}^n$  is defined as rotation by an angle  $\theta = |\theta|$  about the axis parallel to the unit vector  $\hat{\theta} = \theta/|\theta|$ . For example,  $R(\frac{\pi}{2}i)$  rotates the vector by  $\frac{\pi}{2}$  about the x-axis. The matrix representation of the rotation operator in the standard orthonormal basis is given by

$$[R(\theta)] = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Note that the Unitary operators are generalization of Rotation operator in  $\mathbb{V}^n$  which will be seen later.

### Matrix Representation for Product of Operators

Given two linear operators  $A$  and  $B$  acting on a vector space  $V$  over a field  $\mathbb{F}$ , the product of the operators  $AB$  is defined as the operator that results from applying  $A$  followed by  $B$  on any vector  $|v\rangle \in V$ .

$$[AB]_{ij} = \langle i | AB | j \rangle = \sum_{k=1}^n \langle i | A | k \rangle \langle k | B | j \rangle = \sum_{i=1}^n A_{ik} B_{kj} = [A][B]$$

where  $[A]$  and  $[B]$  are the matrix representations of the operators  $A$  and  $B$  respectively. Note that the order of the product of the operators is important and the product of the matrices is not commutative in general. Thus, the matrix representation of the product of the operators  $AB$  is given by the product of the matrix representations of the operators  $A$  and  $B$ .

### Outer Product Representation of Operators

Suppose we are required to represent an operator  $A$  action on input vector space  $V$  and output vector space  $W$  as some linear combination of outer products. Suppose  $A : V \rightarrow W$  is a linear operator with  $|v_i\rangle$  as orthonormal basis for  $V$ , and  $|w_j\rangle$  as orthonormal basis for  $W$ . Thus, we obtain

$$A = I_w A I_v = \sum_{i=1}^n \sum_{j=1}^m |w_j\rangle \langle w_j| A |v_i\rangle \langle v_i| = \sum_{i=1}^n \sum_{j=1}^m \langle w_j | A | v_i \rangle |w_j\rangle \langle v_i|$$

Thus, the operator  $A$  can be represented as a linear combination of outer products of the basis vectors of the input and output vector spaces.

### Properties of Adjoint of an Operator

We know, for a given ket

$$A |v\rangle = |Av\rangle$$

and for the corresponding bra

$$\langle Av| = \langle v| A^\dagger$$

Thus, the adjoint of an operator is defined as

$$(A^\dagger)_{ij} = \langle i| A^\dagger |j\rangle = \langle j| A |i\rangle^* = (A_{ji})^*$$

Also,

$$(AB)^\dagger = B^\dagger A^\dagger$$

#### Example A.1.18.

$$\alpha_1 |V_1\rangle = \alpha_2 |V_2\rangle + \alpha_3 |v_3\rangle \langle V_4|V_5\rangle + \alpha_4 AB |V_6\rangle$$

It's adjoint will be given as:

$$\alpha_1^* \langle V_1| = \alpha_2^* \langle V_2| + \alpha_3^* \langle v_3| \langle V_5|V_4\rangle + \alpha_4^* \langle V_6| B^\dagger A^\dagger$$

In words, when a product of operators, bras, kets and explicit numerical coefficients is encountered, reverse the order of all factors and make the substitutions  $A \rightarrow A^\dagger$  and  $| \rangle \rightarrow \langle |$  and  $\alpha \rightarrow \alpha^*$ .

### A.1.7 Hermitian and Unitary Operators

Note that both the Hermitian and Unitary operators are defined only for a Square Matrices.

#### Hermitian Operators

**Definition A.1.21. Hermitian Operator** is defined as:

$$A^\dagger = A$$

Skew-Hermitian/Anti-Hermitian operator is defined as:

$$A^\dagger = -A$$

#### Properties of Hermitian operators:

- Since  $A^\dagger = A$ , thus the principal diagonal elements will be  $a_{ii} = a_{ii}^*$ , thus the principal diagonal elements of a Hermitian operator are real.
- Principal Diagonal elements for a skew hermitian matrix will satisfy  $a_{ii} = -a_{ii}^*$ , thus the principal diagonal elements of a skew hermitian operator are imaginary.
- Any Matrix can be represented as a sum of Hermitian and skew-Hermitian matrices.

$$A = \frac{A + A^\dagger}{2} + \frac{A - A^\dagger}{2}$$

where the  $\frac{A+A^\dagger}{2}$  is Hermitian and  $\frac{A-A^\dagger}{2}$  is skew-Hermitian.

- Any Complex matrix can be expressed as  $P + iQ$  where P and Q are Hermitian.
- If A and B are Hermitian matrices and commute (i.e.  $AB = BA$ ), then their product is also Hermitian i.e.  $AB$  is Hermitian. Let,

$$(AB)^\dagger = B^\dagger A^\dagger = BA = AB$$

Thus,  $AB$  is Hermitian.

- If A and B are Hermitian then  $A + B$  and  $A - B$  are also Hermitian.

## Unitary Operators

**Definition A.1.22.** Unitary operator is defined as:

$$U^\dagger U = UU^\dagger = I$$

i.e.  $U^\dagger = U^{-1}$

### Properties of Unitary operators:

- Product of two unitary matrix is unitary. Say  $U_1$  and  $U_2$  are unitary matrices, then  $U_1 U_2$  is also unitary.

$$(U_1 U_2)^\dagger U_1 U_2 = U_2^\dagger U_1^\dagger U_1 U_2 = U_2^\dagger U_2 = I$$

- Unitary Operators Preserve inner Products:

$$\langle Uv | Uw \rangle = \langle v | U^\dagger U | w \rangle = \langle v | w \rangle$$

- Unitary Operators Preserve Norms:

$$\|v\| = \sqrt{\langle v|v\rangle}$$

$$\implies \|Uv\| = \sqrt{\langle Uv|Uv\rangle} = \sqrt{\langle v|U^\dagger U|v\rangle} = \sqrt{\langle v|v\rangle} = \|v\|$$

Thus, **Unitary operators preserve inner product and norms and hence are generalizations of rotation operators in  $\mathbb{V}^n$ .** Thus action of a **Unitary operator on any vector just rotates the vector in the vector space and the vector neither scales up or down in norm.**

- Columns of a  $n \times n$  unitary matrix are orthonormal and form a basis for  $\mathbb{C}^n$ . Similarly rows of a Unitary matrix are orthonormal and form a basis for  $\mathbb{C}^n$ .

$$\delta_{ij} = \langle i| I |j\rangle = \langle i| U^\dagger U |j\rangle = \sum_{k=1}^n \langle i| U^\dagger |k\rangle \langle k| U |j\rangle = \sum_{k=1}^n U_{ki}^* U_{kj}$$

Thus, it is 1 only if  $i = j$  and 0 otherwise. Thus, the columns of a unitary matrix are orthonormal. Simliary, it can be proved that the rows of a unitary matrix are orthonormal.

- Determinant of a Unitay Matrix is a complex number of unit modulus.

$$UU^\dagger = I$$

Taking determinant on both sides we get,

$$\det(UU^\dagger) = \det(I)$$

$$\det(U) \det(U^\dagger) = 1$$

$$|\det(U)|^2 = 1$$

$$|\det(U)| = 1$$

Thus, the determinant of a unitary matrix is a complex number of unit modulus.

### Unitarily Diagonizable

**Definition A.1.23.** Unitarily Diagonizable: A matrix  $A$  is said to be unitarily diagonalizable if there exists a unitary matrix  $U$  such that  $U^\dagger AU = D$  where  $D$  is a diagonal matrix. Here,  $U$  is a unitary matrix whose columns are the orthonormal eigen vectors of  $A$ . The matrix  $D$  is a diagonal matrix whose entries are the eigen values of  $A$  corresponding to the eigen vectors of  $A$  written in  $U$ . **Note that the matrix is diagonal in the basis of the eigen vectors of the matrix  $A$ .** Note that any matrix with  $n$  ( $n$  is the dimension of the square matrix) linearly independent Eigen vectors can be written as:

$$AV = V\Lambda$$

where  $V$  is the matrix whose columns are the eigen vectors of  $A$  and  $\Lambda$  is the diagonal matrix whose diagonal elements are the eigen values of  $A$ . Thus, on post-multiplying by  $V^\dagger$  on both sides we get:

$$A = V\Lambda V^\dagger$$

or,

$$V^\dagger AV = \Lambda$$

Thus, the matrix  $A$  is diagonal in its Eigen basis (assuming no degeneracy i.e. it has  $n$  linearly independent eigen vectors.). In the special case when the eigen vectors are orthonormal then the matrix  $V$  is unitary and the matrix  $A$  is unitarily diagonalizable.

### Normal Operators

**Definition A.1.24. Normal Operators:** A normal operator is an operator that commutes with its adjoint. i.e.  $AA^\dagger = A^\dagger A$ .

**Theorem A.1.6.** *An operator is normal if and only if it is unitarily diagonalizable or if a matrix is Unitarily diagonalizable then it is always a normal operator.*

*Proof.* Let  $A$  be Unitarily diagonalizable. Then,

$$A = V\Lambda V^\dagger$$

$$AA^\dagger - A^\dagger A = V\Lambda V^\dagger V\Lambda^\dagger V^\dagger - V\Lambda^\dagger V^\dagger V\Lambda V^\dagger$$

Now since  $V$  is unitary matrix and  $\Lambda$  is a diagonal matrix, thus  $V^\dagger V = I$  and  $\Lambda^\dagger \Lambda = \Lambda \Lambda^\dagger$ . Thus, Substituting we get

$$AA^\dagger - A^\dagger A = V\Lambda\Lambda^\dagger V^\dagger - V\Lambda^\dagger\Lambda V^\dagger = V\Lambda\Lambda^\dagger V^\dagger - V\Lambda\Lambda^\dagger V^\dagger = 0$$

Thus,  $AA^\dagger = A^\dagger A$  and thus A is a normal operator.

Hence if A is unitarily diagonalizable then it is a Normal operator. Now let A be a normal operator. Then,

$$AA^\dagger = A^\dagger A$$

Now we are required to show that A is unitarily diagonalizable. We know that any Matrix A can be decomposed onto a Hermitian and Skew Hermitian matrix as:

$$A = \frac{A + A^\dagger}{2} + \frac{A - A^\dagger}{2}$$

where  $\frac{A+A^\dagger}{2}$  is Hermitian and  $\frac{A-A^\dagger}{2}$  is Skew Hermitian. Now let  $A_H = \frac{A+A^\dagger}{2}$  denote the Skew Hermitian matrix and  $A_{SH} = \frac{A-A^\dagger}{2}$  denote the skew-Hermitian matrix. Thus,  $A = A_H + A_{SH}$ . Now since A is a normal operator we get,

$$AA^\dagger - A^\dagger A = (A_H + A_{SH})(A_H + A_{SH})^\dagger - (A_H + A_{SH})^\dagger(A_H + A_{SH}) = 0$$

$$A_H A_H^\dagger + A_{SH} A_{SH}^\dagger + A_H A_{SH}^\dagger + A_{SH} A_H^\dagger - A_H^\dagger A_H - A_{SH}^\dagger A_{SH} - A_H^\dagger A_{SH} - A_{SH}^\dagger A_H = 0$$

Now substituting the fact that  $A_H \implies A_H = A_H^\dagger$  is Hermitian and  $A_{SH} \implies A_{SH} = -A_{SH}^\dagger$  is Skew Hermitian we get,

$$A_{SH} A_H = A_H A_{SH}$$

Thus,  $A_H$  and  $A_{SH}$  commute. Now since  $A_H$  and  $A_{SH}$  are Hermitian and Skew Hermitian respectively, they are both unitarily diagonalizable. **Using the fact that two matrices commute if and only if they have the same eigen vectors. Thus, both  $A_H$  and  $A_{SH}$  have the same eigen vectors and also the fact that  $A_H$  and  $A_{SH}$  are both unitarily diagonalizable**, thus we get the same unitary matrix V:

$$A_H = V \Lambda_H V^\dagger$$

$$A_{SH} = V \Lambda_{SH} V^\dagger$$

Now  $A = A_H + A_{SH}$ , thus

$$A = V \Lambda_H V^\dagger + V \Lambda_{SH} V^\dagger = V(\Lambda_H + \Lambda_{SH})V^\dagger$$

Now,  $\Lambda_H$  and  $\Lambda_{SH}$  are diagonal matrices, thus  $\Lambda_H + \Lambda_{SH}$  is also a diagonal matrix.

$$A = V \Lambda V^\dagger$$

Thus, A is unitarily diagonalizable where V are the orthonormal eigen vectors of A and  $\Lambda$  is the diagonal matrix with entries as the corresponding eigen values. Hence, proved. **Thus the statement that a matrix is normal if and only if it is unitarily diagonalizable.**  $\square$

### Properties of the Normal operators:

- Hermitian and Unitary operators are both special cases of Normal Operators. Thus, both are unitarily diagonalizable and hence have n linearly independent Eigen vectors.
- A normal matrix is Hermitian if and only if it has real eigen values.

### Positive Operators

**Definition A.1.25. Positive Operators:** A positive operator is an operator such that for any vector  $|v\rangle$  in the vector space, the inner product  $\langle v|A|v\rangle$  is real, non-negative. i.e.  $\langle v|A|v\rangle \geq 0$ .

### Properties of Positive Operators:

- A positive operator is Hermitian.
- For any operator  $A$ ,  $A^\dagger A$  is a positive operator.

$$\langle v|A^\dagger A|v\rangle \geq 0$$

$$\langle v|A^\dagger A|v\rangle = \langle Av|Av\rangle = \langle u|u\rangle \geq 0$$

Since, the inner product of any vector with itself is non-negative.

### A.1.8 Eigen Value Problems

Given a linear operator  $A$  acting on a vector space  $V$  over a field  $\mathbb{F}$ , the eigen value problem is to find the eigen values  $\lambda$  and the corresponding eigen vectors  $|v\rangle$  such that

$$\begin{aligned} A|v\rangle &= \lambda|v\rangle \\ \implies (A - \lambda I)|v\rangle &= 0 \end{aligned}$$

i.e. vectors such that upon the action of the operator  $A$  the vector  $|v\rangle$  is scaled by a factor  $\lambda$  (could be positive or negative, a negative value indicates change in direction and a value less than  $|1|$  indicates scaled down). Thus, the eigen values are the values of  $\lambda$  for which the operator  $A - \lambda I$  has a non-trivial null space. The eigen vectors are the vectors in the null space of the operator  $A - \lambda I$ .

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

This, results in a characteristic equation of the form:

$$P_A(\lambda) = \det(A - \lambda I) = \lambda^n - \beta_1 \lambda^{n-1} + \dots + (-1)^r \beta_r \lambda^{n-r} + \dots + (-1)^n \beta_n = 0$$

In general, we represent eigen value as  $\omega$  and the corresponding eigen vector as  $|\omega\rangle$ .

**Example A.1.19.** For the identitiy operator  $I$ , the eigen value equation is given as:

$$\begin{aligned} I |v\rangle &= \lambda |v\rangle \\ \implies |v\rangle &= \lambda |v\rangle \\ \implies \lambda &= 1 \end{aligned}$$

Thus, the eigen values of the identity operator are  $\lambda = 1$  and the corresponding eigen vectors are all the vectors in the vector space.

**Example A.1.20.** Consider a projection operator  $\mathbb{P}_v$  which projects any vector  $|v\rangle$  onto the subspace spanned by the basis vector  $|v\rangle$ . Thus,  $\mathbb{P}_v |v\rangle = |v\rangle$ . Thus, the eigen value equation is given as:

$$\mathbb{P}_v |v\rangle = \lambda |v\rangle$$

**Case 1:** Consider any ket  $\alpha |v\rangle$ , parallel to  $|v\rangle$  then  $\mathbb{P}_v |v\rangle = |v\rangle = \lambda |v\rangle$ . Thus,  $\lambda = 1$  is an eigen value of the projection operator  $\mathbb{P}_v$ .

**Case 2:** Consider any ket  $\alpha |v_\perp\rangle$ , perpendicular to  $|v\rangle$  then  $\mathbb{P}_v |v_\perp\rangle = 0 = \lambda |v_\perp\rangle$ . Thus,  $\lambda = 0$  is an eigen value of the projection operator  $\mathbb{P}_v$ .

**Case 3:** Consider kets that are neither parallel nor perpendicular to  $|v\rangle$ , then  $\mathbb{P}_v(\alpha |v_\perp\rangle + \beta |v\rangle) = \beta |v\rangle \neq \lambda(\alpha |v_\perp\rangle + \beta |v\rangle)$ . Thus,  $\lambda$  is not an eigen value of the projection operator  $\mathbb{P}_v$ . Thus, we have considered every possible ket in the space and have found all the eigen values and eigen vectors. The eigen values of a projection matrix are  $\lambda = 0$  and  $\lambda = 1$  and their corresponding eigen vectors are the vectors parallel and perpendicular to the subspace spanned by the basis vector  $|v\rangle$  respectively.

**Example A.1.21.** Consider the Rotation operator  $R(\vec{\theta})$  which rotates any vector by an angle  $\frac{\vec{\theta}}{|\vec{\theta}|}$  about the  $\vec{\theta}$  axis. The eigen value equation is given as:

$$R(\vec{\theta}) |v\rangle = \lambda |v\rangle$$

Clearly, any ket parallel to the axis of rotation will not change its direction upon rotation and thus will be an eigen vector of the rotation operator with eigen value  $\lambda = 1$ . Thus, the corresponding eigen vector is  $\vec{\theta}$ . The other two eigen values are  $\lambda = e^{\pm i\theta}$  and the corresponding eigen vectors are the vectors perpendicular to the axis of rotation.

**Properties of Eigen Roots:**

- The sum of the eigen values is equal to the trace of the matrix.

$$Tr(A) = \sum_{i=1}^n a_{ii} = \sum_{i=1}^n \lambda_i = (-1)^{n-1} \beta_{n-1}$$

- The product of the eigen values is equal to the determinant of the matrix.

$$det(A) = \prod_{i=1}^n \lambda_i = (-1)^n \beta_n$$

- if  $det(A) = 0$  then at least one of the eigen value is zero.
- For any triangular matrix, the eigen values are the principal diagonal elements.
- Eigen values of Two Equivalent/Similar matrix are not necessarily same.
- Eigen Values of a Unitary matrix is a complex number with unit modulus and the Eigen vecotrs are mutually orthongonal (Assuming no degeneracy). Let,

$$U |u_i\rangle = u_i |u_i\rangle$$

$$U |u_j\rangle = u_j |u_j\rangle$$

Taking adjoint of the second equation we get:

$$\langle u_j | U^\dagger = \langle u_j | u_j^*$$

Taking inner product with the first equation we get:

$$\langle u_j | U^\dagger U |u_i\rangle = \langle u_j | u_j^* u_i |u_i\rangle$$

$$\langle u_j | u_i\rangle = u_j^* u_i \langle u_j | u_i\rangle$$

$$(1 - (u_j^* u_i)) \langle u_j | u_i\rangle = 0$$

If  $i=j$  then  $\langle u_j | u_i\rangle \neq 0$  thus  $u_i = u_i^*$  i.e. real eigen values. If  $i \neq j$  then  $\langle u_j | u_i\rangle = 0$  i.e. the eigen vectors are orthogonal.

**In case of no degeneracy, Unitary matrix has complex eigen values of unit modulus and othrongonal eigen vectors.**

- If  $\lambda$  is the Eigen root of  $A$  then  $f(\lambda)$  is the Eigen root of  $f(A)$ .
- Suppose  $A$  is a matrix such that  $A^k = 0$  then all the eigen values of  $A$  is zero.
- Complex Eigen Values always exist in pairs.
- Eigen Values of a Hermitian matrix are real. Given  $A$  is a Hermitian matrix,

$$A^\dagger = A$$

$$A|v\rangle = \lambda|v\rangle$$

Taking inner product with  $\langle v|$ , we get:

$$\langle v|A|v\rangle = \lambda\langle v|v\rangle$$

Now adjoint of the equation  $A|v\rangle = \lambda|v\rangle$  is

$$\langle v|A^\dagger = \lambda^*\langle v|$$

Taking inner product with  $|v\rangle$ , we get:

$$\langle v|A^\dagger|v\rangle = \lambda^*\langle v|v\rangle$$

$$\langle v|A|v\rangle = \lambda^*\langle v|v\rangle \quad (\text{since } A \text{ is Hermitian, } A = A^\dagger)$$

Thus, equating the two equations we get:

$$(\lambda - \lambda^*)\langle v|v\rangle = 0$$

Thus,

$$\lambda = \lambda^*$$

Hence, the **Eigen Values of a Hermitian matrix are always real**.

- The Eigen vectors of the Hermitian matrix with respect to distinct eigen values are orthogonal.

Consider the matrix  $A$  with distinct eigen values  $\lambda_i$  and  $\lambda_j$  and the corresponding eigen vectors  $|v_i\rangle$  and  $|v_j\rangle$ . Thus,

$$A|v_i\rangle = \lambda_i|v_i\rangle$$

$$A|v_j\rangle = \lambda_j|v_j\rangle$$

Taking inner product of the above two equations with  $\langle v_j |$  and  $\langle v_i |$  respectively, we get:

$$\langle v_j | A | v_i \rangle = \lambda_i \langle v_j | v_i \rangle$$

$$\langle v_i | A | v_j \rangle = \lambda_j \langle v_i | v_j \rangle$$

Taking adjoint of the above second equations, we get:

$$\langle v_j | A^\dagger | v_i \rangle = \lambda_j^* \langle v_j | v_i \rangle$$

Since A is Hermitian,  $A = A^\dagger$ , thus

$$\langle v_j | A | v_i \rangle = \lambda_j^* \langle v_j | v_i \rangle$$

Thus, equating the two equations, we get:

$$\lambda_i \langle v_j | v_i \rangle = \lambda_j^* \langle v_j | v_i \rangle$$

$$(\lambda_i - \lambda_j^*) \langle v_j | v_i \rangle = 0$$

Thus if  $\lambda_i \neq \lambda_j$  then  $\langle v_j | v_i \rangle = 0$  i.e. the eigen vectors of a Hermitian matrix with respect to distinct eigen values are orthogonal. If  $i = j$  then  $\langle v_j | v_i \rangle \neq 0$  thus,  $\lambda_i = \lambda_j^* \implies \lambda_i = \lambda_i^*$  i.e. real eigen values.

**The Hermitian matrices have all real eigen values and the eigen vectors corresponding to distinct eigen values are orthogonal.**

**Definition A.1.26. Geometric Multiplicity:** The number of linearly independent Eigen vectors associated with a particular  $\lambda$  is called the Geometric multiplicity. Suppose  $\lambda_i$  has geometric multiplicity k  $\implies GM(\lambda_i) = k$  means that k linearly independent eigen vectors exist corresponding to eigen value  $\lambda$ . These vectors form a vector space which is called Eigen Space. In the eigen space all the vectors are only scaled by  $\lambda_i$  and not rotated.

**Definition A.1.27. Algebraic Multiplicity:** The multiplicity of  $\lambda$  as a root of the characteristic equation  $P_A(x) = 0$  is called the algebraic multiplicity. Suppose  $\lambda_i$  has algebraic multiplicity k  $\implies AM(\lambda_i) = k$  means that  $\lambda_i$  is a root of the characteristic equation of multiplicity k.

where n is the dimension of the matrix.

For any matrix A, the algebraic multiplicity of an eigen value is always greater than or equal to the geometric multiplicity.

$$1 \leq AM(\lambda_i) \leq GM(\lambda_i) \leq n$$

Note that any matrix A will have n eigen values (may be repeated i.e.  $AM(\lambda_i) \geq 1$ ), but it is not necessary for it to have n linearly independent eigen vectors. Corresponding to each unique eigen value  $\lambda$  there will be at least one eigen vector. If the  $AM(\lambda_i) > 1$  then there could be more than one linearly independent eigen vector corresponding to the eigen value  $\lambda_i$  but still the number of linearly independent eigen vectors will be less than or equal to the  $AM(\lambda_i)$  i.e.  $GM(\lambda_i) \leq AM(\lambda_i)$ .

**Theorem A.1.7.** *A Hermitian Matrix is always Unitarily Diagonalizable i.e. it has n linearly independent Eigen vectors. ( $AM(\lambda_i)=GM(\lambda_i)$ ) for all  $\lambda_i$ .*

*Proof.* Let A be a Hermitian matrix. Let the geometric multiplicity for some eigen value of A be r ( $< n$ ) where n is the dimension of the matrix. Thus,  $GM(\lambda) = r$ . Thus, we can find r linearly independent eigen vectors (say  $\{|v_1\rangle, |v_2\rangle, \dots, |v_r\rangle\}$ ) corresponding to  $\lambda$ . Let,

$$V^E = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ |v_1\rangle & |v_2\rangle & \dots & |v_r\rangle \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

which is the set of Linearly Independent Orthogonal Eigen vectors. Let,

$$V^\perp = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ |v_{r+1}\rangle & |v_{r+2}\rangle & \dots & |v_n\rangle \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

be the set of orthogonal vectors spanning space orthogonal to  $V^E$ . (Note that  $V^E$  and  $V^\perp$  are orthogonal subspaces to each other). Let,

$$V = [V^E \ V^\perp] = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ |v_1\rangle & |v_2\rangle & \dots & |v_n\rangle \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Now  $A|v_i\rangle = \lambda|v_i\rangle$ , for  $i = 1, 2, \dots, r$ . Thus,

$$AV = \begin{bmatrix} \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \lambda|v_1\rangle & \lambda|v_2\rangle & \dots & \lambda|v_r\rangle & A|v_{r+1}\rangle & \dots & A|v_n\rangle \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

Now, a vectors in  $\mathbb{C}^n$  can be written as a linear combination of the columns of V. This,  $Av_i = \sum_{j=1}^m c_{ij}v_j$  for  $i = r+1, r+2, \dots, n$ . Thus, in matrix form we can write this as:

$$A \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ |v_1\rangle & |v_2\rangle & \dots & |v_r\rangle & |v_{r+1}\rangle & \dots & |v_n\rangle \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} = V \begin{bmatrix} \lambda_1 & & & c_{r+1,1} & \dots & c_{n,1} \\ \lambda_2 & & & c_{r+1,2} & \dots & c_{n,2} \\ \lambda_3 & & & c_{r+1,3} & \dots & c_{n,3} \\ \ddots & & & \vdots & \vdots & \vdots \\ & & & \lambda_r & c_{r+1,r} & \dots & c_{n,r} \\ 0 & 0 & 0 & \dots & 0 & c_{r+1,r+1} & \dots & c_{n,r+1} \\ 0 & 0 & 0 & \dots & 0 & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & c_{r+1,n} & \dots & c_{n,n} \end{bmatrix}$$

$$AV = VM$$

Thus, pre-multiplying by  $V^\dagger$  on both sides we get:

$$V^\dagger AV = V^\dagger VM$$

Now,  $V^\dagger V = I$  since the columns of V are orthonormal ( $V^E$  and  $V^\perp$  are orthogonal subspaces, we have chosen the vectors in  $V^E$  and  $V^\perp$  such that they are orthonormal). Thus,

$$V^\dagger AV = M$$

Now taking adjoint on both sides we get:

$$V^\dagger A^\dagger V = M^\dagger$$

Using the fact that A is Hermitian, we get:

$$V^\dagger AV = M^\dagger$$

Thus,  $M = M^\dagger$  i.e. M is Hermitian. Thus, M can be written as:

$$M = \begin{bmatrix} \lambda & 0 & \dots & 0 \\ \lambda & 0 & \dots & 0 \\ \lambda & 0 & \dots & 0 \\ \ddots & \vdots & \vdots & \vdots \\ & \lambda & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & c_{r+1,r+1} & \dots & c_{n,r+1} \\ 0 & 0 & 0 & \dots & 0 & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & c_{r+1,n} & \dots & c_{n,n} \end{bmatrix}$$

Thus, let the matrix C be:

$$C = \begin{bmatrix} c_{r+1,r+1} & \dots & c_{n,r+1} \\ \vdots & \vdots & \vdots \\ c_{r+1,n} & \dots & c_{n,n} \end{bmatrix}$$

Since, M is a Hermitian Matrix thus C must also be a Hermitian Matrix. Now, since  $M = V^\dagger AV = V^{-1}AV$ , thus M and A are similar matrices. Thus, M and A must have the same Eigen values and Characteristic Polynomial. Thus,

$$P_M(z) = P_A(z)$$

$$(\lambda - z)^r \det(C - zI) = \det(A - zI)$$

Thus, now we need to show that  $\lambda$  is not an eigen value of C thus proving that  $AM(\lambda) = GM(\lambda) = r$  for matrix A. This is done using proof by contradiction. Let  $\lambda$  be a root of  $\det(C - zI)$ . Thus,

$$Cu = \lambda u$$

where  $u \in \mathbb{C}^{m-r}$  is an eigen vector of C. Thus, we can write  $\hat{u} \in \mathbb{C}^m$  as:

$$\hat{u} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ u_{r+1} \\ \vdots \\ u_n \end{bmatrix}$$

Now,

$$V^\dagger AV\hat{u} = M\hat{u} = \begin{bmatrix} \lambda & & 0 & \dots & 0 \\ & \lambda & 0 & \dots & 0 \\ & & \lambda & \dots & 0 \\ & & & \ddots & \vdots \\ & & & & \lambda & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & c_{r+1,r+1} & \dots & c_{n,r+1} \\ 0 & 0 & 0 & \dots & 0 & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & c_{r+1,n} & \dots & c_{n,n} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ u_{r+1} \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \lambda u_{r+1} \\ \vdots \\ \lambda u_n \end{bmatrix}$$

$$\implies \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \lambda u_{r+1} \\ \vdots \\ \lambda u_n \end{bmatrix} = \lambda \begin{bmatrix} 0 \\ \vdots \\ 0 \\ u_{r+1} \\ \vdots \\ u_n \end{bmatrix} = \lambda \hat{u}$$

Thus,

$$V^\dagger A V \hat{u} = \lambda \hat{u}$$

$\hat{u}$  is an eigen vector of  $V^\dagger A V$  with eigen value  $\lambda$ . Premultiplying by  $V$  on both sides we get:

$$AV\hat{u} = \lambda V\hat{u}$$

Hence,  $V\hat{u}$  is an eigen vector of  $A$  corresponding to  $\lambda$ . But,  $V\hat{u}$  lies in the vector space of  $\{|v_{r+1}\rangle, \dots, |v_n\rangle\}$  which contradicts the assumption that  $\{|v_1\rangle, \dots, |v_r\rangle\}$  are the only linearly independent eigen vectors of  $A$  corresponding to  $\lambda$ . Thus,  $\lambda$  is not an eigen value of  $C$ . **Thus, a Hermitian Matrix is always Unitarily diagonalizable. A similar proof can be used to show that a Skew Hermitian Matrix is also always Unitarily diagonalizable.**  $\square$

### Spectral Decomposition

**Theorem A.1.8.** *Any normal operator  $M$  on a vector space  $V$  is diaongal with respect to some orthonormal basis for  $V$ . Conversely, any diagonalizable operator is normal.*

This means that  $M$  can be written as a sum of orthogonal projection operators onto the eigenspaces of  $M$  where  $|\lambda_i\rangle$  are eigen vectors of  $M$ .

$$M = \sum_{i=1}^n \lambda_i \mathbb{P}_i$$

where  $\lambda_i$  are the eigen values of  $M$  and  $\mathbb{P}_i = |\lambda_i\rangle \langle \lambda_i|$  are the orthogonal projection operators onto the eigenspaces of  $M$ . This can also be realised as the fact that any normal operator is always unitarily diagonalizable and hence can be written as an outer product representation. Thus,

$$M = V \Lambda V^\dagger = \sum_{i=1}^n \lambda_i |\lambda_i\rangle \langle \lambda_i| = \sum_{i=1}^n \lambda_i \mathbb{P}_i$$

where  $V$  is the matrix of orthogonal eigen vectors,  $\Lambda$  is a diagonal marix of corresponding eigen values.

### Commutator and Anti-Commutator

**Definition A.1.28.** The commutator of two operators  $A$  and  $B$  is defined as:

$$[A, B] = AB - BA$$

The anti-commutator of two operators  $A$  and  $B$  is defined as:

$$\{A, B\} = AB + BA$$

### Properties of the Commutator

- $AB = \frac{[A, B] + \{A, B\}}{2}$
- if  $[A, B] = 0, \{A, B\} = 0$  and  $A$  is invertible then  $B = 0$ .
- $[A, B]^\dagger = [B^\dagger, A^\dagger]$
- $[A, B] = -[B, A]$
- If  $A$  and  $B$  are Hermitian then  $\iota[A, B]$  is Hermitian.

### Simultaneous Diagonalization of two Hermitian Operators

**Definition A.1.29.** Suppose  $A$  and  $B$  are Hermitian operators, Then  $[A, B] = 0$  if and only if there exists an orthonormal basis such that both  $A$  and  $B$  are diagonal with respect to that basis. We can say that  $A$  and  $B$  are simultaneously diagonalizable.

### Operator Functions

We restrict ourselves to functions that can be written as a power series. Consider a series

$$f(x) = \sum_{n=0}^{\infty} a_n x^n$$

where  $a_n$  are complex numbers. Then we can define the same function for an operator to be:

$$f(A) = \sum_{n=0}^{\infty} a_n A^n$$

where  $A^n$  is the product of the operator  $A$  with itself  $n$  times.

**Example A.1.22.** Consider the exponential series:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \sum_{i=0}^{\infty} \frac{x^n}{n!}$$

Then the exponential of an operator A is defined as:

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \dots = \sum_{i=0}^{\infty} \frac{A^n}{n!}$$

Now we restrict ourselves to Hermitian operator A, then in the eigen basis of A we can write:

$$A = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}$$

Thus, the  $A^n$  in the exponential series will be given as:

$$A^n = \begin{pmatrix} \lambda_1^n & 0 & \dots & 0 \\ 0 & \lambda_2^n & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n^n \end{pmatrix}$$

Thus, the exponential of the Hermitian operator A will be given as:

$$e^A = \begin{pmatrix} \sum_{i=1}^{\infty} \frac{\lambda_1^n}{n!} & 0 & \dots & 0 \\ 0 & \sum_{i=1}^{\infty} \frac{\lambda_2^n}{n!} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sum_{i=1}^{\infty} \frac{\lambda_n^n}{n!} \end{pmatrix}$$

$$e^A = \begin{pmatrix} e^{\lambda_1} & 0 & \dots & 0 \\ 0 & e^{\lambda_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & e^{\lambda_n} \end{pmatrix}$$

Some results:

- $e^{A^\dagger} = (e^A)^\dagger$
- $e^{A+B} = e^A e^B$  if  $[A, B] = 0$
- $e^{A^\dagger} e^A = I$
- $e^{\iota A}$  is unitary if A is Hermitian and  $\det(e^{\iota A}) = e^{\iota \text{Tr}(H)}$ .

### Trace of a Matrix

**Definition A.1.30.** The trace of a matrix is the sum of the diagonal elements of the matrix. It is denoted by  $Tr(A)$ .

$$Tr(A) = \sum_{i=1}^n a_{ii}$$

### Properties of Trace

- $Tr(AB) = Tr(BA)$
- $Tr(A + B) = Tr(A) + Tr(B)$
- $Tr(A^\dagger) = Tr(A)$
- If  $tr(AA^\dagger) = 0$  then  $A = 0$  (Null matrix)
- $\|v\|^2 = Tr(vv^\dagger)$
- $Tr(ABC) = Tr(BCA) = Tr(CAB)$

### A.1.9 Norms

**Definition A.1.31.** A real valued function defined on a vector space ( $\mathbb{C}^m \rightarrow \mathbb{R}$ ) is called a norm (denoted by  $\|x\|$ ) if it satisfies the following properties:

- $\|x\| \geq 0$  and  $\|x\| = 0$  if and only if  $x = 0$ .
- $\|\alpha x\| = |\alpha| \|x\|$  for any scalar  $\alpha$ .
- $\|x + y\| \leq \|x\| + \|y\|$  (Triangle Inequality).

Thus,

$$\|\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n\| \leq |\alpha_1| \|x_1\| + |\alpha_2| \|x_2\| + \cdots + |\alpha_n| \|x_n\|$$

where  $\alpha_i$  are scalars and  $x_i$  are vectors.

In general, the norm of a vector is the measure of the length of the vector. The p-norm of a vector is defined as:

$$\|x\|_p = \left( \sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}}$$

where  $x = (x_1, x_2, \dots, x_n)$  is a vector in  $\mathbb{C}^n$ . Here,  $|x_i|$  is the modulus of the complex number  $x_i$ .

The 1-norm of a vector is defined as:

$$\|x\|_1 = \sum_{i=1}^n |x_i|$$

The 2-norm of a vector is called the Euclidean norm and is defined as:

$$\|x\|_2 = \left( \sum_{i=1}^n |x_i|^2 \right)^{\frac{1}{2}}$$

The  $\infty$ -norm of a vector is defined as:

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$$

The  $\infty$ -norm is the maximum of the absolute values of the elements of the vector.

### Properties of Norms

- $\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1$
- $\|x\|_2 \leq \sqrt{n} \|x\|_\infty$

where  $x \in \mathbb{C}^n$ .

### A.1.10 Induce Matrix Norm

**Definition A.1.32.** The induced matrix norm of a matrix  $A$  is defined as:

$$\|A\|_p = \max_{\|v\|_p=1} \|Av\|_p$$

where  $\|v\|_p$  is p-the norm of the vector v.

In other words, it is the factor by which a matrix can scale a vector. It is also called Spectral Norm. For the purpose of this notes, we will consider the induced 2-norm i.e. the Euclidean norm for the vectors.

### Properties of Induced Matrix Norm

- $\|A\| = \max_{\|v\|=1} \|Av\| = \max_{\|v\|=1} \sqrt{\langle Av | Av \rangle} = \max_{\|v\|=1} \sqrt{\langle v | A^\dagger A | v \rangle}$

- For a diagonal matrix  $D$ ,  $\|D\|_1 = \|D\|_2 = \max_{1 \leq i \leq n} |d_i|$  where  $d_i$  are the diagonal elements of the matrix  $D$ .
- For any matrix  $A$ ,  $\|A\|_1 = \text{maximum absolute column sum of the matrix } A$ .
- For any matrix  $A$ ,  $\|A\|_\infty = \text{maximum absolute row sum of the matrix } A$ .
- For any matrix  $A$ ,  $\|A\|_2 = \text{maximum singular value of the matrix } A$ .
- For any matrix  $A$ ,  $\|A\|_2 = \sqrt{\rho(A^\dagger A)}$  where  $\rho(A^\dagger A)$  is the maximum eigen value of the matrix  $A^\dagger A$ .
- For any matrix  $A$ ,  $\|Ax\| \leq \|A\|\|x\|$ . This is because  $\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}$ .
- Sub-Multiplicative Property:  $\|AB\| \leq \|A\|\|B\|$  Let  $\|x\| = 1$  then,

$$\|AB\| = \sup_{\|x\|=1} \|ABx\|$$

Now, using the property that  $\|Ax\| \leq \|A\|\|x\|$  we get:

$$\|ABx\| \leq \|A\|\|Bx\| \leq \|A\|\|B\|$$

Now, for some  $x = \hat{x}$ ,  $\|AB\hat{x}\| = \|AB\|$ . Thus,  $\|AB\| \leq \|A\|\|B\|$ .

- $\|A + B\| \leq \|A\| + \|B\|$
- For a Unitary matrix  $U$ ,  $\|U\| = 1$ . Thus,  $\|UA\|_2 = \|A\|_2$ . This is because Unitary Matrix preserves norm of a vector (it only rotates the vectors).

### A.1.11 Conditioning and Condition Number

In this section we find the condition number of a matrix times vector multiplication.

Let  $y = Ax$  where  $A$  is matrix and  $x, b$  are vectors. The input given is a vector

and the output is  $Ax$ . Thus,

$$\begin{aligned}
 \kappa(x) &= \max_{\delta x} \frac{\frac{\|A(x+\delta x) - Ax\|}{\|Ax\|}}{\frac{\|\delta x\|}{\|x\|}} \\
 &= \max_{\delta x} \frac{\|A\delta x\|}{\|Ax\|} \frac{\|x\|}{\|\delta x\|} \\
 &= \max_{\delta x} \frac{\|A\delta x\|}{\|\delta x\|} \frac{\|x\|}{\|Ax\|} \\
 &= \frac{\|x\|}{\|Ax\|} \left( \max_{\delta x} \frac{\|A\delta x\|}{\|\delta x\|} \right) \\
 &= \frac{\|x\|}{\|Ax\|} \|A\| \\
 &= \frac{\|A\| \|x\|}{\|Ax\|}
 \end{aligned}$$

Let  $A$  be a square matrix and non-singular. Then  $A^{-1}$  exists, thus,  $x = A^{-1}Ax \implies \|x\| = \|A^{-1}Ax\| \leq \|A^{-1}\| \|Ax\|$ . Thus,  $\frac{\|x\|}{\|Ax\|} \leq \|A^{-1}\|$ . Substituting this, in the previous result, we get,

$$\kappa(x) \leq \|A\| \|A^{-1}\|$$

Thus, we define the condition number for a matrix vector multiplication  $Ax$  with  $\kappa(x) = \|A\| \|A^{-1}\|$ . Building upon this, consider the problem of solving the system of linear equations  $Ax = b$  by using  $x = A^{-1}b$ . Then the condition number will be given as  $\kappa(b) = \|A^{-1}\| \|A\|$ . Thus, we define the condition number of a matrix  $A$  as  $\kappa(A) = \|A\| \|A^{-1}\|$ . In the 2-norm sense,

$$\kappa(A) = \|A\|_2 \|A^{-1}\|_2 = \frac{\sigma_{max}}{\sigma_{min}} = \sqrt{\frac{\lambda_{max}(A^\dagger A)}{\lambda_{min}(A^\dagger A)}}$$

where  $\sigma_{max}$  and  $\sigma_{min}$  are the maximum and minimum singular values respectively. If  $A$  is Hermitian then the condition number is given as:

$$\kappa(A) = \frac{|\lambda_{max}(A)|}{|\lambda_{min}(A)|}$$

Note that here  $\lambda_{max}$  and  $\lambda_{min}$  are in the absolute sense.

### A.1.12 Overview of Iterative Methods

These methods dominate the large-scale computations. The importance of iterative algorithms in linear algebra stems from the fact that non-iterative or direct algorithms for general matrices require  $\mathcal{O}(m^3)$  computations. It is very high when  $m$  is large, and in the relative sense that since the input to most matrix problem involves only  $\mathcal{O}(m^2)$  inputs. Large matrices arise indirectly in the discretization of differential or integral equations. For very very large  $m$ , it is an approximation to  $\infty$ . They have some kind of structure, and ways have been found to exploit this structure in more and more contexts. The most obvious structure that a large matrix may have is sparsity i.,e. preponderance of zero entries. For example, a finite difference discretization of the partial differential equation may lead to a matrix of dimension  $m = 10^5$  with only  $\nu = 10$  nonzero entries per row. This kind of structure is readily exploited by the iterative methods we shall discuss.

For these algorithms we use a matrix in the form of a black box. Given input  $x$ , it outputs  $Ax$ . Some iterative algorithm may require the computation of  $A^*x$ . The iterative algorithm requires nothing more than the ability to determine  $Ax$  for any  $x$ , which in a computer program will be effected by a procedure whose internal workings need be of no concern to the designer of the iterative algorithm. For the example of a sparse matrix  $A$ , it is easy to design a procedure to compute  $Ax$  in only  $\mathcal{O}(\nu m)$  rather than  $\mathcal{O}(m^2)$  operations. most recently, it has become apparent that other kinds of matrix structure may also be exploitable, even though the matrices involved are dense. For example, the solution of integral equations by numerical methods typically leads to dense matrix problems; in engineering these are called boundary element or panel methods. The coefficients have a deal of regularity in them. All this is in marked contrast to the algorithms of direct linear algebra, such as Gaussian or Householder triangularization, which explicitly manipulate matrix entries so as to introduce zeros, but in the process generally destroy sparsity. Sparsity has been the kind of structure exploited most often in iterative matrix computations.

#### Krylov Subspaces

The idea is of projecting an  $m$ -dimensional problem into a lower-dimensional Krylov subspace. Given a matrix  $A$  and a vector  $b$ , the associated Krylov sequence is the set of vectors  $b, Ab, A^2b, A^3b, \dots$ , which can be computed by the black box in the form  $b, Ab, A(Ab), A(A(Ab)), \dots$ . The corresponding Krylov subspaces are the spaces spanned by successively larger group of these vectors.

### A.1.13 Conclusions

- A Normal operators ( $AA^\dagger = A^\dagger A$ ) are always unitarily diagonalizable (i.e.  $A = U^\dagger D U$  where  $U$  is a Unitary matrix whose columns are orthonormal eigen vectors and  $D$  is a diagonal matrix made up of eigen values of the matrix  $A$ .) Thus, any normal matrix has orthogonal (linearly independent) eigen vectors (n is the dimension of the matrix).
- Hermitian matrix and Unitary matrix are special cases of Normal operators. Thus, the Hermitian matrix is unitarily diagonalizable and has real eigen values and n orthogonal linearly independent eigen vectors. The Unitary matrix is also unitarily diagonalizable and has complex eigen values of unit modulus and n orthogonal linearly independent eigen vectors. Since both the Hermitian and Unitary matrices are unitarily diagonalizable since they are normal, thus both of them have spectral decomposition. Thus both Unitary and Hermitian matrix can be written as a sum of orthogonal projection operators onto the eigenspaces of the matrix as shown:

$$M = \sum_{i=1}^n \lambda_i |\lambda_i\rangle\langle\lambda_i| = \sum_{i=1}^n \lambda_i \mathbb{P}_i$$

where  $\lambda_i$  are the eigen values of M and  $|\lambda_i\rangle$  are the eigen vectors of M. Here,  $\mathbb{P}_i = |\lambda_i\rangle\langle\lambda_i|$  are the orthogonal projection operators onto the eigenspaces of M.

- If two matrices commute then they have the same eigen vectors (note that the eigen values could be same but still the matrix would have the n linearly independent eigen vectors). Let,  $AB = BA$  then,

$$\begin{aligned} A|v\rangle &= \lambda|v\rangle \\ \implies BA|v\rangle &= \lambda B|v\rangle \\ \implies AB|v\rangle &= \lambda B|v\rangle \quad (\text{Since } AB=BA) \end{aligned}$$

Let  $B|v\rangle = |u\rangle$  then,

$$A|u\rangle = \lambda|u\rangle$$

Thus  $|u\rangle$  is an eigen vector of A. Thus,  $|u\rangle = B|v\rangle$  is an eigen vector of A corresponding to  $\lambda$  (assuming no degeneracy). Thus,  $B|v\rangle = p|v\rangle$ . Thus,  $|v\rangle$  is an eigen vector of B with the eigen value corresponding to p.

## A.2 Tensor Products

It is a method for combining two vector spaces to obtain a new larger vector space.

**Definition A.2.1.** Suppose  $V$  and  $W$  are vector spaces of dimensions  $m$  and  $n$  respectively. For convinence we also suppose that  $V$  and  $W$  are Hilbert Spaces. Then,  $V \otimes W$  (read as  $V$  tensor  $W$ ) is a vector space of dimension  $mn$  such that any vector in  $V \otimes W$  can be written as a linear combination of the tensor products of the vectors in  $V$  and  $W$ . i.e The elements of  $V \otimes W$  are linear combination of 'tensor products'  $|v\rangle \otimes |w\rangle$  of elements  $|v\rangle$  of  $V$  and  $|w\rangle$  of  $W$ . For example, if  $|i\rangle$  and  $|j\rangle$  are orthonormal bases for the spaces  $V$  and  $W$  then  $|i\rangle \otimes |j\rangle$  is a basis fr  $V \otimes W$ . Note that we often use abbreviated notations  $|v\rangle |w\rangle$ ,  $|v, w\rangle$  or  $|vw\rangle$  for the tensor product  $|v\rangle \otimes |w\rangle$ .

- For an arbitrary scalar  $z$  and elements of  $|v\rangle$  of  $V$  and  $|w\rangle$  of  $W$ .

$$z(|v\rangle \otimes |w\rangle) = (z|v\rangle) \otimes |w\rangle = |v\rangle \otimes (z|w\rangle)$$

- For arbitrary  $|v_1\rangle$  and  $|v_2\rangle$  in  $V$  and  $|w\rangle$  in  $W$ ,

$$(|v_1\rangle + |v_2\rangle) \otimes |w\rangle = |v_1\rangle \otimes |w\rangle + |v_2\rangle \otimes |w\rangle$$

- For arbitrary  $|v\rangle$  in  $V$  and  $|w_1\rangle$  and  $|w_2\rangle$  in  $W$ ,

$$|v\rangle \otimes (|w_1\rangle + |w_2\rangle) = |v\rangle \otimes |w_1\rangle + |v\rangle \otimes |w_2\rangle$$

**Example A.2.1.** if  $V$  is a two-dimensional vector space with basis vectors  $|0\rangle$  and  $|1\rangle$  then  $|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle$  is an element of  $V \otimes V$ .

Linear operators acting on the space  $V \otimes W$ . Suppose  $|v\rangle$  and  $|w\rangle$  are vectors in  $V$  and  $W$ , and  $A$  and  $B$  are linear operators on  $V$  and  $W$  respectively. Then we can deined a linear operator  $A \otimes B$  on  $V \otimes W$  by the equation

$$(A \otimes B)(|v\rangle \otimes |w\rangle) = A|v\rangle \otimes B|w\rangle$$

It can be shown that  $A \otimes B$  is a well-defined linear operator on  $V \otimes W$ . The definition of  $A \otimes B$  is then exteneded to all the elements of  $V \otimes W$  in the natural way to ensure linearity of  $A \otimes B$ , that is,

$$(A \otimes B)\left(\sum_i a_i |v_i\rangle \otimes |w_i\rangle\right) = \sum_i a_i A|v_i\rangle \otimes B|w_i\rangle$$

Consider the case where  $A : V \rightarrow V'$  and  $B : W \rightarrow W'$  are linear operators on the vector spaces  $V$  and  $W$ . Then, an arbitrary linear operator  $C$  mapping  $V \otimes W$  to  $V' \otimes W'$  can be represented as a linear combination of tensor products of operators mapping  $V$  to  $V'$  and  $W$  to  $W'$ . That is,

$$C = \sum_i c_i A_i \otimes B_i$$

where by definition,

$$\left( \sum_i c_i A_i \otimes B_i \right) |v\rangle \otimes |w\rangle = \sum_i c_i A_i |v\rangle \otimes B_i |w\rangle$$

The inner product on the spaces  $V$  and  $W$  can be used to define a natural inner product on  $V \otimes W$ . Define

$$\left( \sum_i a_i |v_i\rangle \otimes |w_i\rangle, \sum_j b_j |v'_j\rangle \otimes |w'_j\rangle \right) \equiv \sum_{ij} a_i * b_j \langle v_i | v'_j \rangle \langle w_i | w'_j \rangle$$

It can be shown that the function so defined is a well-defined inner product. The inner product space  $V \otimes W$  inherits the other structure we are familiar with such as notions of an adjoint, unitary, normality and Hermiticity.

### Kronecker Product

Suppose  $A$  is a  $m$  by  $n$  matrix, and  $B$  is a  $p$  by  $q$  matrix. Then we have the matrix representation:

$$A \otimes B \equiv \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{pmatrix}$$

In this representation terms like  $a_{11}B$  denote  $p$  by  $q$  matrices whose entries are proportional to  $B$ , with overall proportionality constant  $a_{11}$ .

**Useful notation:**  $|\psi\rangle^{\otimes n}$  denotes  $\psi$  tensored with itself  $k$  times. For example,  $|\psi\rangle^{\otimes 2} = \psi \otimes \psi$ . Similar notation can be used for operators/matrices.

### Properties of Tensor Products:

- Transpose, Complex conjugation and adjoint operations distributed over the tensor product,

$$(A \otimes B)^T = A^T \otimes B^T$$

$$(A \otimes B)^* = A^* \otimes B^*$$

$$(A \otimes B)^\dagger = A^\dagger \otimes B^\dagger$$

- Tensor product of two unitary operators is unitary.
- Tensor product of two positive operators is positive.
- Tensor product of two projectors is a projector.

**Example A.2.2.** Consider one qubit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ . Then the tensor product of  $|\psi\rangle$  with itself is:

$$|\psi\rangle \otimes |\psi\rangle = (\alpha|0\rangle + \beta|1\rangle) \otimes (\alpha|0\rangle + \beta|1\rangle) = \alpha^2|00\rangle + \alpha\beta|01\rangle + \alpha\beta|10\rangle + \beta^2|11\rangle$$

Thus, in matrix form:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} \otimes \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha & \alpha \\ \beta & \beta \end{pmatrix} = \begin{pmatrix} \alpha^2 \\ \alpha\beta \\ \alpha\beta \\ \beta^2 \end{pmatrix}$$

**Example A.2.3.** Consider the matrix  $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$  and  $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$ . Then the Kronecker product of A and B is:

$$A \otimes B = \begin{pmatrix} 1 \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} & 2 \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \\ 3 \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} & 4 \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{pmatrix}$$

**Example A.2.4.** Consider  $|\psi\rangle = \iota|0\rangle + 7|1\rangle$  and  $|\phi\rangle = |00\rangle + 3|10\rangle + 7|11\rangle$ . Then, the tensor product  $|\psi\rangle \otimes |\phi\rangle = |\psi\rangle |\phi\rangle = |\psi\phi\rangle = (\iota|0\rangle + 7|1\rangle)(|00\rangle + 3|10\rangle + 7|11\rangle) = \iota|0\rangle|00\rangle + 3\iota|0\rangle|10\rangle + 7\iota|0\rangle|11\rangle + 7|1\rangle|00\rangle + 21|1\rangle|10\rangle + 49|1\rangle|11\rangle = \iota|000\rangle + 3\iota|010\rangle + 7\iota|011\rangle + 7|100\rangle + 21|110\rangle + 49|111\rangle$ . Thus, in matrix form:

$$\begin{pmatrix} \iota \\ 7 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \\ 3 \\ 7 \end{pmatrix} = \begin{pmatrix} \iota & 0 & 3 & 7 \\ 7 & 0 & 3 & 7 \\ 7 & 1 & 0 & 3 \\ 7 & 3 & 7 & 0 \end{pmatrix} = \begin{pmatrix} \iota \\ 0 \\ 3\iota \\ 7\iota \\ 7 \\ 0 \\ 21 \\ 49 \end{pmatrix}$$

**Example A.2.5.** Say  $|\alpha\rangle = 3|0\rangle + \iota|1\rangle$  and  $|\beta\rangle = |00\rangle + 2|10\rangle + 7|11\rangle$  the the outer product  $|\alpha\rangle\langle\beta|$  is given as:

$$\begin{aligned} |\alpha\rangle\langle\beta| &= (3|0\rangle + \iota|1\rangle)(\langle 00| + 2\langle 10| + 7\langle 11|) \\ &= 3|0\rangle\langle 00| + 6|0\rangle\langle 10| + 21|0\rangle\langle 11| + \iota|1\rangle\langle 00| + 2\iota|1\rangle\langle 10| + 7\iota|1\rangle\langle 11| \end{aligned}$$

In matrix form, it can be written as:

$$\begin{pmatrix} 3 & 0 & 6 & 21 \\ \iota & 0 & 2\iota & 7\iota \end{pmatrix}$$

**Example A.2.6.** Say a matrix  $A = \begin{pmatrix} 0 & 1 \\ 3 & \iota \\ 7 & 0 \\ 0 & 13 \end{pmatrix}$ . Then in the outer product form it

can be written as:

$$A = |00\rangle\langle 1| + 3|01\rangle\langle 0| + \iota|01\rangle\langle 1| + 7|10\rangle\langle 0| + 13|11\rangle\langle 1|$$

### Factorizing Tensor Products

Consider a  $|\psi\rangle = \frac{1}{2}|00\rangle - \frac{\iota}{2}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$ , then we can factorize the tensor product by following the following procedure. Assume that the  $|\psi\rangle = |\phi\rangle|\chi\rangle$  where  $|\phi\rangle = a|0\rangle + b|1\rangle$  and  $|\chi\rangle = c|0\rangle + d|1\rangle$  are vectors in the spaces V and W respectively. Then we can write:

$$|\psi\rangle = |\phi\rangle\otimes|\chi\rangle = |\phi\rangle|\chi\rangle = |\phi\chi\rangle = (a|0\rangle+b|1\rangle)(c|0\rangle+d|1\rangle) = ac|00\rangle+ad|01\rangle+bc|10\rangle+bd|11\rangle$$

Now equating it with  $|\psi\rangle$  we get:

$$|\psi\rangle = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle = \frac{1}{2}|00\rangle - \frac{\iota}{2}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

Thus,

$$ac = \frac{1}{2}, ad = 0, bc = -\frac{\iota}{2}, bd = \frac{1}{\sqrt{2}}$$

Thus,  $a = \frac{1}{\sqrt{2}}, b = 0, c = 1, d = \frac{1}{\sqrt{2}}$ . Thus, the tensor product can be factorized as  $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle|1\rangle + \frac{1}{\sqrt{2}}|1\rangle|1\rangle$ .

Consider a  $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle$ , then we can factorize the tensor product by foolowing the following procedure. Assume that the  $|\psi\rangle = |\phi\rangle|\chi\rangle$  where  $|\phi\rangle =$

$a|0\rangle + b|1\rangle$  and  $|\chi\rangle = c|0\rangle + d|1\rangle$  are vectors in the spaces V and W respectively. Then we can write:

$$|\psi\rangle = |\phi\rangle \otimes |\chi\rangle = |\phi\rangle |\chi\rangle = |\phi\chi\rangle = (a|0\rangle + b|1\rangle)(c|0\rangle + d|1\rangle) = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle$$

Now equating it with  $|\psi\rangle$  we get:

$$|\psi\rangle = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle$$

Thus,

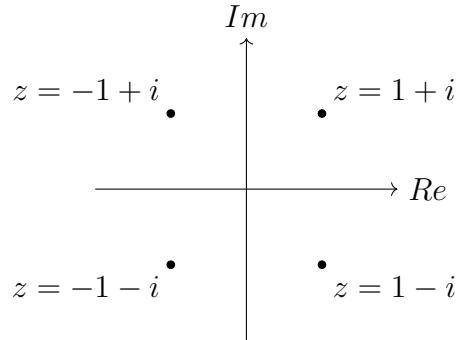
$$ac = \frac{1}{\sqrt{2}}, ad = 0, bc = 0, bd = \frac{1}{\sqrt{2}}$$

Thus, no such values of a,b, c and d exist. Hence, it cannot be factorized.

## A.3 Complex Numbers

### A.3.1 Complex Plane

The complex plane is as denoted in the figure below:



With y axis as imaginary axis, and x axis as real axis. A complex number in cartesian coordinate form is denoted as  $z = a + \imath b$  where  $\imath = \sqrt{-1}$  and  $a, b \in \mathbb{R}$  which can be written as a column vector  $z = \begin{pmatrix} a \\ b \end{pmatrix}$ .

### A.3.2 Addition of Complex Numbers

The addition of complex numbers is as follows:

$$z_1 + z_2 = (a_1 + \imath b_1) + (a_2 + \imath b_2) = (a_1 + a_2) + \imath(b_1 + b_2)$$

Thus, the addition of complex numbers is the same as the addition of vectors in the complex plane.

**Properties:**

- Commutative:  $z_1 + z_2 = z_2 + z_1$
- Associative:  $(z_1 + z_2) + z_3 = z_1 + (z_2 + z_3)$
- Identity:  $z + 0 = z$  where  $0 = 0 + 0\iota$
- Inverse:  $z + (-z) = 0$
- Distributive:  $z_1(z_2 + z_3) = z_1z_2 + z_1z_3$

### A.3.3 Multiplication of Complex Numbers

The multiplication of complex numbers is as follows:

$$z_1z_2 = (a_1 + \iota b_1)(a_2 + \iota b_2) = a_1a_2 - a_1b_2 + \iota(a_1b_2 + a_2b_1)$$

Thus, the multiplication of complex numbers is the same as the multiplication of vectors in the complex plane.

**Properties:**

- Commutative:  $z_1z_2 = z_2z_1$
- Associative:  $(z_1z_2)z_3 = z_1(z_2z_3)$
- Identity:  $z1 = 1z = z$
- Distributive:  $z_1(z_2 + z_3) = z_1z_2 + z_1z_3$
- Scalar Multiplication:  $\alpha(z_1z_2) = \alpha z_1 z_2$
- Scalar Multiplication:  $(\alpha + \beta)z = \alpha z + \beta z$
- Scalar Multiplication:  $\alpha(\beta z) = (\alpha\beta)z$
- Existence of multiplicative identity:  $1z = z$
- Existence of Multiplicative inverse:  $z \neq 0 \implies \frac{1}{z} = \frac{z^*}{z\bar{z}}$

### A.3.4 Complex Conjugate

The complex conjugate of a complex number  $z = a + \imath b$  is denoted by  $\bar{z} = a - \imath b$ . The complex conjugate of a complex number is the reflection of the complex number about the real axis.

**Properties:**

- $(z_1 + z_2)^* = z_1^* + z_2^*$
- $(z_1 z_2)^* = z_1^* z_2^*$
- $(z^*)^* = z$
- $z + z^* = 2\operatorname{Re}(z)$
- $z - z^* = 2i\operatorname{Im}(z)$
- $zz^* = |z|^2$
- $\frac{1}{z} = \frac{z^*}{zz^*}$
- $z^{-1} = \frac{z^*}{|z|^2}$

### A.3.5 Modulus of a Complex Number

The modulus of a complex number  $z = a + \imath b$  is denoted by  $|z| = \sqrt{a^2 + b^2}$ . The modulus of a complex number is the distance of the complex number from the origin in the complex plane.

**Properties:**

- $|z|^2 = zz^*$
- $|z_1 z_2| = |z_1| |z_2|$
- $|z_1 + z_2| \leq |z_1| + |z_2|$
- $|z_1 - z_2| \geq ||z_1| - |z_2||$
- $|z_1 - z_2|^2 = |z_1|^2 + |z_2|^2 - 2\operatorname{Re}(z_1 z_2^*)$
- $|z_1 + z_2|^2 = |z_1|^2 + |z_2|^2 + 2\operatorname{Re}(z_1 z_2^*)$

- $|z_1 + z_2|^2 + |z_1 - z_2|^2 = 2(|z_1|^2 + |z_2|^2)$
- $|z_1 + z_2|^2 + |z_1 - z_2|^2 = 4|z_1|^2|z_2|^2$
- $|z_1 + z_2|^2 + |z_1 - z_2|^2 = 4|z_1 z_2|^2$
- $|z_1 + z_2|^2 + |z_1 - z_2|^2 = 4|z_1|^2|z_2|^2$

### A.3.6 Polar form

The polar form of a complex number  $z = a + \imath b$  is denoted by  $z = r(\cos(\theta) + \imath \sin(\theta))$  where  $r = |z|$  and  $\theta = \tan^{-1}(\frac{b}{a})$ . The polar form of a complex number is the representation of the complex number in terms of its modulus and argument.

**Properties:**

- $z = r(\cos(\theta) + \imath \sin(\theta)) = r(\cos(\theta + 2\pi n) + \imath \sin(\theta + 2\pi n))$  where n is an integer.
- $z_1 z_2 = r_1 r_2 (\cos(\theta_1 + \theta_2) + \imath \sin(\theta_1 + \theta_2))$
- $\frac{1}{z} = \frac{1}{r} (\cos(-\theta) + \imath \sin(-\theta))$
- $z^n = r^n (\cos(n\theta) + \imath \sin(n\theta))$

### A.3.7 Euler's Formula

$z = a + \imath b = re^{\imath\theta}$ . Recall that the expansion of  $e^x$  using taylor series can be written as:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \sum_{i=0}^{\infty} \frac{x^n}{n!}$$

Now substituting  $\imath\theta$  in place of x, we get:

$$e^{\imath\theta} = 1 + \imath\theta + \frac{(\imath\theta)^2}{2!} + \frac{(\imath\theta)^3}{3!} + \dots = \sum_{i=0}^{\infty} \frac{(\imath\theta)^n}{n!}$$

$$e^{\imath\theta} = 1 + \imath\theta - \frac{\theta^2}{2!} - \imath\frac{\theta^3}{3!} + \dots = \cos(\theta) + \imath \sin(\theta)$$

On rearranging the terms we get:

$$e^{\imath\theta} = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \dots + \imath(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \dots)$$

Now recall that the expansion of sin and cos using taylor series expansion are:

$$\sin(\theta) = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \dots = \sum_{i=0}^{\infty} \frac{(-1)^i \theta^{2i+1}}{(2i+1)!}$$

$$\cos(\theta) = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \dots = \sum_{i=0}^{\infty} \frac{(-1)^i \theta^{2i}}{(2i)!}$$

Thus, Euler's formula states that  $e^{i\theta} = \cos(\theta) + i \sin(\theta)$ . Hence, on substituting this in the polar form of a complex number we get:

$$z = a + ib = \sqrt{a^2 + b^2} \left( \cos \left( \tan^{-1} \frac{b}{a} \right) + i \sin \left( \tan^{-1} \frac{b}{a} \right) \right) = r(\cos(\theta) + i \sin(\theta)) = re^{i\theta}$$

where  $r = |z|$  and  $\theta = \tan^{-1} \left( \frac{b}{a} \right)$ . Using, this we can also write:

$$\cos \theta = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

$$\sin \theta = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$

### A.3.8 Roots of Unity

The nth roots of unity are the solutions to the equation  $z^n = 1$  (recall that in the complex numbers, there exist n solutions to the equation, for example, if  $n=2$ , z could be 1 or -1. If  $n=4$ , z could be  $1, i, -1, -i$ ). These roots can be written as powers of  $\omega = e^{2\pi i/n}$ , which is called the primitive nth root of unity. The nth roots of unity are given by:

$$\omega_n^k = e^{\frac{2\pi k i}{n}}$$

where k is an integer such that  $0 \leq k \leq n - 1$ . The nth roots of unity are equally spaced around the unit circle in the complex plane (it can be seen that  $\omega$  lies on the units circle so  $|\omega| = 1$ , and the angle made by the root  $\omega$  is the angle  $\phi = 2\pi/n$ , and squaring  $\omega$ , we double the angle. thus, raising to the jth power,  $\omega^j$  has a phase angle  $\phi = 2j\pi/n$  and is still an nth root of unity.).

#### Properties:

- $\omega_n^k = \omega_n^{(k \bmod n)}$  where  $k \bmod n$  is the remainder when k is divided by n. For example, the 4th roots of unity are  $\omega_4^0 = e^{\frac{2\pi i 0}{4}} = 1, \omega_4^1 = e^{\frac{2\pi i 1}{4}} = i, \omega_4^2 = e^{\frac{2\pi i 2}{4}} = -1, \omega_4^3 = e^{\frac{2\pi i 3}{4}} = -i$ . Note: The roots are periodic with a period of n. Here,  $n = 4$ .

### A.3.9 Discrete Fourier Transform Matrix

The Discrete Fourier Transform (DFT) matrix is a square matrix whose entries are the  $n$ th roots of unity. The  $i,j$ th element of the matrix is given by  $\omega_n^{ij}$ . The DFT matrix of size  $n$  is given by:

$$F_n = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}$$

The DFT matrix is a unitary matrix, that is,  $F_n^\dagger F_n = I$ . The DFT matrix is used to perform the Discrete Fourier Transform which is used in signal processing, image processing, data compression, etc.

**Example A.3.1.** Transform  $|\psi\rangle = \frac{|0\rangle + |3\rangle}{\sqrt{2}}$  using  $F_4$ .

$$\begin{aligned} F_4 &= \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \iota & -1 & \iota \\ 1 & -1 & 1 & -1 \\ 1 & \iota & -1 & \iota \end{pmatrix} \\ |\psi\rangle &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\ F_4 |\psi\rangle &= \frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \iota & -1 & \iota \\ 1 & -1 & 1 & -1 \\ 1 & \iota & -1 & \iota \end{pmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{2\sqrt{2}} \begin{bmatrix} 2 \\ 1-\iota \\ 0 \\ 1+\iota \end{bmatrix} \\ &= \frac{2|0\rangle + (1-\iota)|1\rangle + (1+\iota)|3\rangle}{2\sqrt{2}} \end{aligned}$$

**Example A.3.2.** Write  $QFT_2$ . Because  $n = 2, \omega_2 = e^{\frac{2\pi\iota}{2}} = -1$ . Thus, the QFT matrix of size 2 is given by:

$$QFT_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & \omega \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

which is clearly equal to  $H$ , the hadamard matrix. Write  $QFT_4$ . Because  $n = 4$ ,  $\omega_4 = e^{\frac{2\pi i}{4}} = i$ . Thus, the QFT matrix of size 4 is given by:

$$QFT_4 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

which is clearly equal to  $H^{\otimes 2}$ .

**Example A.3.3.** Find the quantum fourier transform for  $n = 4$  of the functions

$$|f\rangle = \frac{1}{2}(|0\rangle + |1\rangle + |2\rangle + |3\rangle) = \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad |g\rangle = |0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ and } |h\rangle = |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}.$$

Their corresponding QFT are given by:

$$QFT_4 |f\rangle = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 4 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$QFT_4 |g\rangle = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix}$$

$$QFT_4 |h\rangle = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ i \\ -1 \\ -i \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.5i \\ -0.5 \\ -0.5i \end{pmatrix}$$

In examples, note that the columns of  $QFT_4$  are orthogonal i.e. their inner product is zero and along with that the 2-norm of the columns is 1. Thus, the QFT matrix is unitary.

Note that the vectors like  $|f\rangle$  that had fewer zeros (narrow spread) had fourier transforms with a lot of zeros (larger spread), and vice-versa.

Finally, in examples, notice how the only difference between the foueir transforms of  $|g\rangle$  and  $|h\rangle$  is a difference of realtive phase shifts.

Thus, the fourier transform takes the vector  $\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{n-1} \end{pmatrix}$  to the vector  $\begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{n-1} \end{pmatrix}$  where

$\beta_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \omega_n^{kj} \alpha_j$ . Quantum Fourier transform on measurement we would get one of the values of this superposition with probability  $|\alpha|^2$  where  $\alpha$  is the coefficient of the state in the superposition. The quantum Fourier transform is used in quantum algorithms such as Shor's algorithm for factoring large numbers and the quantum phase estimation algorithm for estimating the eigenvalues of unitary operators.

### Properties of QFT

- **QFT is unitary**

*Proof.* To prove that QFT is a unitary matrix we are required to prove that any two columns of QFT matrix are orthonormal i.e.

$$\langle C_i | C_j \rangle = \delta_{ij}$$

where  $C_i$  and  $C_j$  are the columns of the QFT matrix.

Let  $C_i$  and  $C_j$  be the ith and jth columns of the QFT matrix. Then, the inner product of the two columns is:

$$\begin{aligned} \langle C_i | C_j \rangle &= \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{-ik} (\omega_n^j)^k \\ &= \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{(j-i)k} \end{aligned}$$

If  $i = j$ , then the sum is equal to n i.e.  $\langle C_i | C_j \rangle = 1$ . If  $i \neq j$ , then the sum is equal to zero. Here, for the case  $i \neq j$  it forms a geometric series. Recall, that the sum of a geometric series is given by  $S_n = \frac{a(r^n - 1)}{r - 1}$  where a is the first term, r is the common ratio and n is the number of terms. Here  $a = 1, r = \omega^{i-j}, n = n$ . Thus, the sum of the geometric series is:

$$S_n = \frac{\omega_n^{(j-i)n} - 1}{\omega_n^{j-i} - 1}$$

Using the property that  $\omega_n^{ln} = 1$ , for some integer l. Thus, the sum of the geometric series is:

$$S_n = \frac{1 - 1}{\omega_n^{j-i} - 1} = 0$$

Thus, the inner product of the two columns is zero. Thus, the QFT matrix is a unitary matrix.

**because, the Fourier transform is a unitary operator, we can implement it in a quantum circuit. Thus is  $N = 2^n$ , we can apply the Fourier transform  $QFT_N = QFT_{2^n}$  to a n-qubit system.**  $\square$

- **Linear Shift:** QFT Matrix when acts on a linearly shifted state vector causes a phase shift in its Fourier transform. Mathematically  $|f(x)\rangle$ ,  $x \in \mathbb{Z}_{2^n}$  has Fourier transform  $|\hat{f}(x)\rangle$ , then  $|f(x + j)\rangle$  has Fourier transform  $|\hat{f}(x)\rangle e^{\frac{2\pi x j}{2^n}}$ . Also since  $QFT_{2^n}$  is unitary and  $QFT_{2^n}QFT_{2^n}^\dagger = I$ , the converse is true. A linear phase shift on  $|f\rangle$  produces a linear shift in  $ket\hat{f}(x)$ . Consider the following QFT Matrix of size n:

$$F_n = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}$$

Let  $|\psi\rangle = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{pmatrix}$  be a vector of size n. Then, the QFT of  $|\psi\rangle$  is given by:

Say, when  $F_n$  acts on  $|\psi\rangle$  we get,

$$F_n |\psi\rangle = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{n-1} \end{pmatrix}$$

So now both input and output vectors are in superposition state. Thus, on measuring the output we get,  $|k\rangle$  with probability  $|\beta_k|^2$ . Now, we linearly shift

the vector  $|\psi\rangle$  by say 1 position. Thus, the new vector will be as shown:

$$|\psi'\rangle = \begin{pmatrix} \psi_{n-1} \\ \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-3} \\ \psi_{n-2} \end{pmatrix}$$

Now, when  $F_n$  acts on  $|\psi'\rangle$  we get,

$$F_n |\psi'\rangle = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} \psi_{n-1} \\ \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-3} \\ \psi_{n-2} \end{pmatrix} = \begin{pmatrix} \omega_n^0 \beta_0 \\ \omega_n^1 \beta_1 \\ \omega_n^2 \beta_2 \\ \vdots \\ \omega_n^{n-2} \beta_{n-2} \\ \omega_n^{n-1} \beta_{n-1} \end{pmatrix}$$

Thus, upon measurement we get  $|k\rangle$  with probability  $|\omega^k \beta_k|^2 = |\omega^k|^2 |\beta_k|^2 = |\beta_k|^2$ , since the modulus of  $|\omega_k| = 1$ . Thus, the QFT matrix when acts on a linearly shifted vector introduces a phase shift in the output.

*Proof.* This can be shown as follows: Consider the  $\beta_k$  before the linear shift is:

$$\beta_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \omega_n^{jk} \psi_j$$

Now, after linear shifting input by  $m$  we get:

$$\beta'_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \omega_n^{jk} \psi_{j-m} = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \omega_n^{k(j+m)} \psi_j$$

Now we are required to show that this output is the same as the previous output but with a phase shift to it. Thus, upon multiplying the  $\beta_k$  by  $\omega_n^{mk}$  we get:

$$\omega_n^{mk} \beta_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \omega_n^{(j+m)k} \psi_j = \beta'_k$$

Thus, we proved that the QFT matrix when acts on a linearly shifted vector introduces a phase shift in the output.  $\square$

**Example A.3.4.** For  $N = 2^2 = 4$ , let  $|\Theta\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix}$  and  $|\Phi\rangle = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_0 \end{pmatrix}$ .

$$QFT_4 |\Theta\rangle = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix}$$

$$QFT_4 |\Phi\rangle = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_0 \end{pmatrix} = \begin{pmatrix} \beta_0 \\ -\iota\beta_1 \\ -\iota\beta_2 \\ \iota\beta_3 \end{pmatrix}$$

The only difference between  $|\Theta\rangle$  and  $|\Phi\rangle$  is a relative phase shift. Note that it doesn't matter if we are going to measure a state, then the phases don't matter at all, because if the phase is  $e^{i\theta}$ , then upon measurement  $|e^{i\theta}| = 1$ . Therefore the phase of a given state does not effect the probability of measuring that state. In order to gather information about the phases, consider the following

example. Consider the states  $|\Theta\rangle = \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$  and  $|\Phi\rangle = \frac{1}{2} \begin{pmatrix} 1 \\ \iota \\ -1 \\ -\iota \end{pmatrix}$ . We can't tell the difference by measuring the states. Upon applying  $QFT_4$ . We get:

$$QFT_4 |\Theta\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$QFT_4 |\Phi\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Thus, measuring the Fourier Transform of the states will reveal the relative phases.

- **Period and Wavelength Relationship:**

Consider a periodic function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  with a period  $r$  which takes

n bits as input and n bits as output. Now we apply a Quantum Fourier Transformation on it which is Discrete Fourier Transform.

$$QFT_{2^n} f(x) = \hat{f}(x)$$

where  $\hat{f}(x)$  is the Fourier Transform of the function f. Now, the period of the function f is the smallest positive integer r such that  $f(x) = f(x + r)$  for all x. This is called the period of the function f and is denoted by r. The wavelength of the Fourier Transform of the function f is the smallest positive integer s such that  $\hat{f}(x) = \hat{f}(x + s)$  for all x. The wavelength of the Fourier Transform of the function f is denoted by s. The period and wavelength of the function f are related by the equation:

$$s = \frac{2^n}{r}$$

where n is the number of bits in the input/output of the function f. Thus, in matrix form we can write it as:

$$\frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_r \\ \psi_0 \\ \psi_1 \\ \vdots \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{\frac{2^n}{r}-1} \\ \beta_0 \\ \beta_1 \\ \vdots \end{pmatrix}$$

This, can be as shown in the figure A.1.

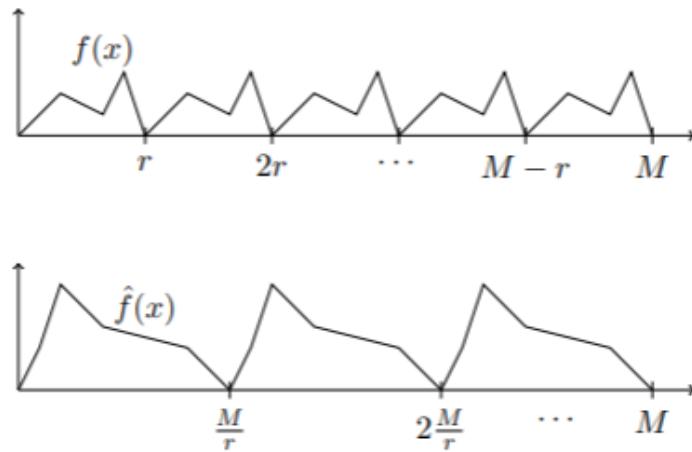


Figure A.1: Period and Wavelength Relationship

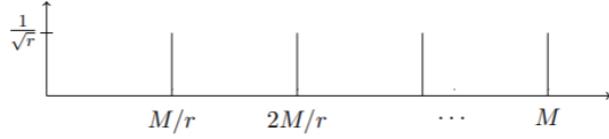
In general, the wider the range of a function, the sharper the range in the Fourier domain; and vice versa. In example, the fourier transform of a delta function is an even spread, while the transform of an even spread is a delta function. Consider the case where,

$$f(x) = \begin{cases} \sqrt{\frac{r}{2^n}} & \text{if } x = 0 \pmod{r} \\ 0 & \text{otherwise} \end{cases}$$

This is what is used in Shor's algorithm.

*Proof.* Consider the following figure A.2 and A.3 ( $M = 2^n$ ).

Figure A.2:  $f(x)$

Figure A.3: QFT of  $f(x)$ 

Suppose  $|\alpha\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{2^n-1} \end{pmatrix}$  has the Fourier transform  $|\beta\rangle = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{2^n-1} \end{pmatrix}$ , then the  $j$ th component of its Fourier trasnform is given by (recall that  $n$  is the number of Qubits):

$$\beta_j = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \omega_{2^n}^{jk} \alpha_k$$

Now in our case,

$$f(x) = \begin{cases} \sqrt{\frac{r}{2^n}} & \text{if } x = 0 \pmod{r} \\ 0 & \text{otherwise} \end{cases}$$

Thus, we calculate the Fourier Transform as:

$$\hat{f}(x) = \frac{\sqrt{r}}{2^n} \sum_{i=0}^{\frac{2^n}{r}-1} \omega^{rix}$$

where we have written  $\hat{f}(x)$  instead of  $\hat{f}(j)$ , because the  $\hat{f}(j)$  is equal to the Fourier transform evaluated at  $x = j$ . Using the formula for summation of geometric series. We get:

$$\hat{f}(x) = \frac{\sqrt{r}}{2^n} \sum_{i=0}^{\frac{2^n}{r}-1} \omega^{rix} = \frac{\sqrt{r}}{2^n} \frac{1 - \omega^{x2^n}}{1 - \omega^{rx}}$$

Recall that  $\omega^{2^n j} = 1$  (because  $\omega^{2^n} = 1$ ), so that the numerator is always equal to 0. So the only time the denominator is equal to zero is when  $rx = k2^n$ , or  $x = \frac{k2^n}{r} \equiv 0 \pmod{\frac{M2^n}{r}}$ . In this case, the numerator and denominator are both 0, so we must compute the limit using the l'Hospitals rule.

$$\lim_{x \rightarrow \frac{k2^n}{r}} \frac{1 - \omega^{x2^n}}{1 - \omega^{rx}} = \lim_{x \rightarrow \frac{k2^n}{r}} \frac{-2^n \omega^{x2^n-1}}{-r \omega^{rx-1}} = \lim_{x \rightarrow \frac{k2^n}{r}} \frac{2^n \omega^{x2^n}}{r \omega^{rx}} = \frac{2^n}{r}$$

Pluggin this result back, the outcome is

$$\hat{f}(x) = \begin{cases} \frac{1}{\sqrt{r}} & \text{if } x = 0 \pmod{\frac{2^n}{r}} \\ 0 & \text{otherwise} \end{cases}$$

This property can also be proced in general, imagin a periodic function in r of this tupe s a basis for any periodic function. Allow the possibility of relative phase shifts and we can thus prove this property in general.  $\square$

### A.3.10 Classical Fast Fourier Transform

The Fourier transform is an  $N \times N$  matrix, straightforward multiplication by  $F_N$  takes  $\mathcal{O}(N^2)$  stpes to carry out, because multiplication of f on each row takes  $N$  multiplications. The FFT reduced this to  $\mathcal{O}(N \log N)$  steps. The FFT required only that  $N = 2^n$  for some integer n, but this is a relatively easy requirement becuase the computers can simply choose their domain.

The fast Fourier transform uses the symmetry of the Fourier transform to reduce the computation time. We rewrite the Fourier transform of size  $n$  as two fourier transforms of size  $N/2$  - the odd and the even terms. We then repeat this over and over agina to exponentially reduce the time. To see this work in detail, the matrix of the Fourier transform consider the  $QFT_8$ :

$$QFT_8 = \frac{1}{\sqrt{8}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega \end{pmatrix}$$

Note that how row j and j+4 are similar and how column j and j+4 are similar. Thus, we split the fourier transform up nito even and odd columns. Consider the following figure A.4:

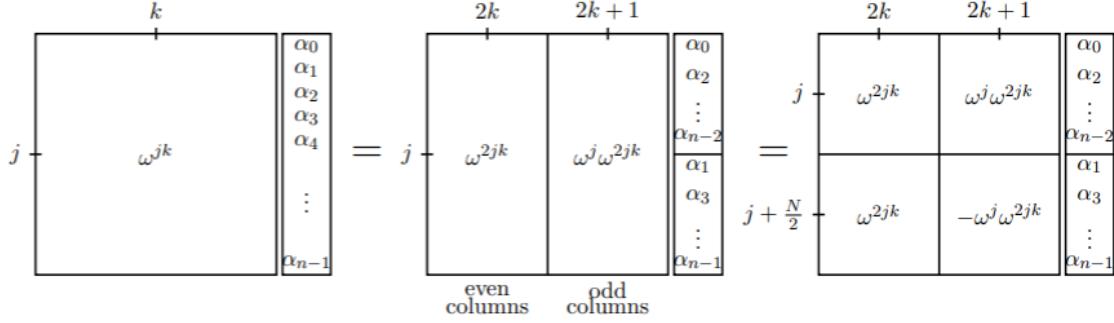


Figure A.4: FFT of size 8

In the first frame, we have represented the whole Fourier transform matrix, by describing the  $j$ th row and  $k$ th column:  $\omega^{jk}$ . In the next frame, we separate the odd and even columns, and similarly separate the vector that is to be transformed. In the third frame, we add a little symmetry by noticing that  $\omega^{j+N/2} = -\omega^j$  (since  $\omega^{N/2} = -1$ ).

Notice that both the odd and even sides contain the term  $\omega^{2jk}$ . But if  $\omega$  is the primitive  $N$ th root of unity, then  $\omega^2$  is the primitive  $N/2$ nd root of unity. Therefore, the matrices whose entries are  $j,k$ th entry is  $\omega^{2jk}$  are really just  $QFT_{N/2}$ . Now we can write  $QFT_N$  in a new way:

Now suppose we are calculating the Fourier transform of the function  $f(x)$ . We can write the above manipulations as an equation that computes the  $j$ th term  $\hat{f}(j)$ .

$$\hat{f}(j) = (\vec{F}_{2^{n/2}} f_{even})(j) + \omega^j (\vec{F}_{2^{n/2}} f_{odd})(j)$$

This turns our calculation of  $QFT_N$  into two applications of  $QFT_{N/2}$ . We can turn this into four applications of  $QFT_{N/4}$ , and so forth. As long as  $N = 2^n$  for some  $n$ , we can break down our calculation of  $QFT_N$  into  $\log N$  steps, each of which takes  $\mathcal{O}(N)$  steps. Thus, the FFT takes  $\mathcal{O}(N \log N)$  steps to compute the Fourier transform of a function of size  $N$ .

## A.4 Probability Theory

## A.5 Calculus

# Appendix B

## Classical Computations

The Turing Machine is an abstract model of computation that can perform any computation that can be performed by a digital computer. It is a mathematical model of computation that defines an abstract machine which manipulates symbols on a strip of tape according to a table of rules. The machine operates on an infinite memory tape divided into discrete cells. Each cell contains a symbol from a finite alphabet. The machine has a read/write head that can read the symbol on the tape and write a new symbol. The machine can move the tape left or right one cell at a time. The machine has a finite set of states and a table of rules that determine the next state of the machine based on the current state and the symbol read from the tape. The machine halts when it reaches a halting state. The Turing Machine can perform any computation that can be performed by a digital computer. It is a universal model of computation that can simulate any other computational model.

But in real world, the computers are finite in size. An alternative model of computation is the circuit model which is equivalent to the Turing Machine and is suitable for studying the complexity of algorithms. The circuit model consists of gates that perform logical operations on bits. The gates are connected by wires that carry the bits from one gate to another. The circuit model can be used to represent any computation that can be performed by a digital computer. The circuit model is a useful abstraction that allows us to reason about the efficiency of algorithms and design better algorithms.

**Definition B.0.1. Logic Gate:** It is a function defined as  $f : \{0, 1\}^k \rightarrow \{0, 1\}^l$  where  $k$  is the number of input bits and  $l$  is the number of output bits.

Note that no loops are allowed in such circuits to avoid instabilities. Thus, they are Acyclic circuits. This is the Circuit Model of Computation. Some of the useful Logic Gates are:

## B.1 Classical Logic Gates

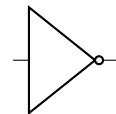
### B.1.1 NOT Gate

The NOT gate is a unary gate that takes a single input bit and produces a single output bit. The output bit is the complement of the input bit i.e. if the input bit is 0, the output bit is 1 and if the input bit is 1, the output bit is 0. The truth table for the NOT gate is:

Input	Output
0	1
1	0

Table B.1: Truth table for NOT Gate

The NOT gate is denoted by the symbol  $\neg$ . Its symbol is as shown:



Mathematically, the NOT gate is defined as:

$$f(x) = \neg x$$

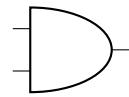
### B.1.2 AND Gate

The AND gate is a binary gate that takes two input bits and produces a single output bit. The output bit is 1 if both input bits are 1 and 0 otherwise. The truth table for the AND gate is:

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Table B.2: Truth table for AND Gate

The AND gate is denoted by the symbol  $\wedge$ . Its symbol is as shown:



Mathematically, the AND gate is defined as:

$$f(x, y) = x \wedge y$$

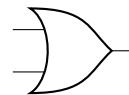
### B.1.3 OR Gate

The OR gate is a binary gate that takes two input bits and produces a single output bit. The output bit is 1 if at least one of the input bits is 1 and 0 otherwise. The truth table for the OR gate is:

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

Table B.3: Truth table for OR Gate

The OR gate is denoted by the symbol  $\vee$ . Its symbol is as shown:



Mathematically, the OR gate is defined as:

$$f(x, y) = x \vee y$$

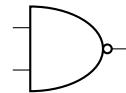
### B.1.4 NAND Gate

The NAND gate is a binary gate that takes two input bits and produces a single output bit. The output bit is the complement of the AND gate i.e. the output bit is 0 if both input bits are 1 and 1 otherwise. The truth table for the NAND gate is:

Input 1	Input 2	Output
0	0	1
0	1	1
1	0	1
1	1	0

Table B.4: Truth table for NAND Gate

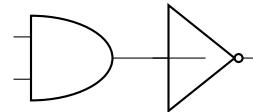
The NAND gate is denoted by the symbol  $\overline{\cdot}$ . Its symbol is as shown:



Mathematically, the NAND gate is defined as:

$$f(x, y) = \neg(x \wedge y)$$

Note that the NAND Gate can also be written by adding a NOT gate at the output of the AND gate as shown:



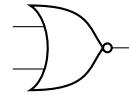
### B.1.5 NOR Gate

The NOR gate is a binary gate that takes two input bits and produces a single output bit. The output bit is the complement of the OR gate i.e. the output bit is 0 if at least one of the input bits is 1 and 1 otherwise. The truth table for the NOR gate is:

Input 1	Input 2	Output
0	0	1
0	1	0
1	0	0
1	1	0

Table B.5: Truth table for NOR Gate

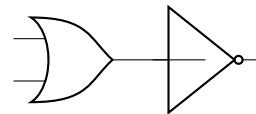
The NOR gate is denoted by the symbol  $\overline{A \vee B}$  Its symbol is as shown:



Mathematically, the NOR gate is defined as:

$$f(x, y) = \neg(x \vee y)$$

NOR Gate can also be written by adding a NOT gate at the output of the OR gate as shown:



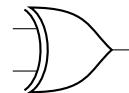
### B.1.6 XOR Gate

The XOR gate is a binary gate that takes two input bits and produces a single output bit. The output bit is 1 if the input bits are different and 0 otherwise. The truth table for the XOR gate is:

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Table B.6: Truth table for XOR Gate

The XOR gate is denoted by the symbol  $\oplus$ . Its symbol is as shown:



Mathematically, the XOR gate is defined as:

$$f(x, y) = x \oplus y$$

### B.1.7 FANOUT Gate

The FANOUT gate is a unary gate that takes a single input bit and produces two output bits. The output bits are the same as the input bit. The truth table for the FANOUT gate is:

Input	Output 1	Output 2
0	0	0
1	1	1

Table B.7: Truth table for FANOUT Gate

Mathematically, the FANOUT gate is defined as:

$$f(x) = (x, x)$$

### B.1.8 CROSSOVER Gate

The CROSSOVER gate is a binary gate that takes two input bits and produces two output bits. The output bits are the complement of the input bits i.e. if the input bits are (0,1), the output bits are (1,0) and if the input bits are 1, the output bits are (0,1). The truth table for the CROSSOVER gate is:

Input 1	Input 2	Output 1	Output 2
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

Table B.8: Truth table for CROSSOVER gate

This is same as saying that the value of two bits are interchanged.

Note that OR gate can be simulated by three NOT gates and one AND gate using **De Morgan's law**

$$a \vee b = \neg(\neg a \wedge \neg b)$$

Similarly XOR gate can be simulated by AND and NOT gates as

$$a \oplus b = (\neg a \wedge b) \vee (a \wedge \neg b)$$

which can be further simplified by using the above definition of OR gate using AND and NOT gate.

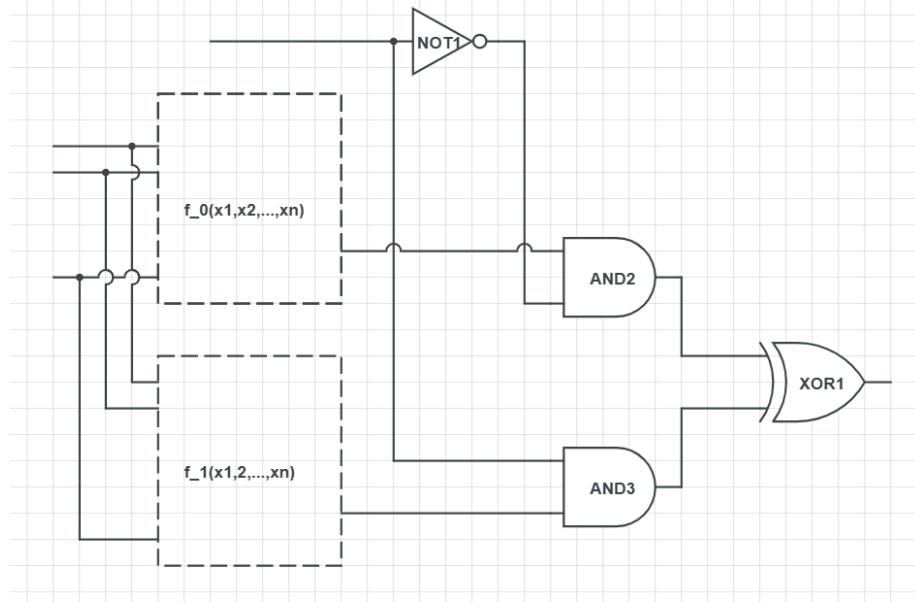
## B.2 Classical Circuits

**Definition B.2.1. Ancilla bits:** The Ancilla or work bits are the extra bits that are used in the computation to store the intermediate results. They are temporary storage devices that are used to store the intermediate results and are not a part of the inputs or the output bits of the computation.

**Theorem B.2.1.** *There always exists a classical circuit which can simulate any classical boolean function  $f$ .*

*Proof.* Any fixed gates can be used to compute any function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ . Consider the proof for the simplified function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  with  $n$  input bits and a single output bit. This is a Boolean function. Using mathematical induction, For  $n = 1$ , there are four possible functions: the identity, which is a single wire; the bit flip, implemented using a NOT Gate; the function which replaces the input bit with a 0, which can be obtained using ANDing the input with a work bit initially in the 0 state; and the function which replaces the input bit with a 1, which can be obtained using ORing the input with a work bit initially in the 1 state. Thus, we have proved for  $n=1$ .

Now for some  $n$ , suppose that any function on  $n$  bits may be computed by a circuit and let  $f$  be a function on  $n + 1$  bits. Define  $f_0$  and  $f_1$  to be the functions on  $n$  bits obtained by fixing the last bit of the input to 0 and 1 respectively. So  $f_0 = f(x_1, x_2, \dots, x_n, 0)$  and  $f_1 = f(x_1, x_2, \dots, x_n, 1)$ . By the induction hypothesis, there are circuits  $C_0$  and  $C_1$  that compute  $f_0$  and  $f_1$  respectively. Thus, now we are required to design a circuit with computes  $f$ . Thus, we design a circuit that computes both  $f_0$  and  $f_1$  on the first  $n$  bits and then depending on whether the first bit of the input was 0 or a 1 it outputs the appropriate answer. This can be done using a circuit as shown in figure B.1.

Figure B.1: Circuit to compute  $f$ 

□

For the Universal Circuit Construction:

- wires, which preserve the states of the bits;
- ancilla bits prepared in the standard states, used in the  $n=1$  case of the proof;
- FANOUT operation, which takes a single bit as input and outputs two copies of that bit;
- The CROSSOVER operation, which interchanges the values of the two bits;
- the AND, OR, and NOT operations, which are the basic logic gates.

In the case of quantum wires for the preservation of qubits qubit state, it is not necessarily obvious that good quantum wires for the preservation of qubits can be constructed. The FANOUT operation cannot be performed because of the **No - cloning Theorem**. The AND and XOR Gates are not invertible thus irreversible and hence can't be implemented in a straight forward manner as unitary quantum gates. Thus, implementing a universal quantum circuit is not as straight forward as in the classical case.

**Example B.2.1.** Half adder and Full adder circuits used in Classical Computers.

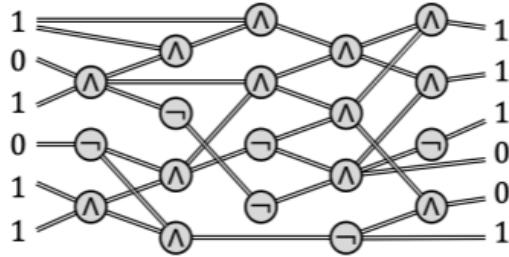


Figure B.2: Generic form of a classical circuit (information flows from left to right)

## B.3 Classical Algorithms as logic circuits

We can represent algorithms as logic circuits with the inputs on the left, time flows left to right and the outputs are on the right. Classical computer algorithms usually described in high level languages, but they implicitly represent many low-level logic operations, like this circuit shown in figure B.2 of example. A reasonable overall cost measure is the number of gates. There are other measures that we may care about such as the *depth*, which is the length of the longest path from an input to an output. Or the *width*, which is the maximum number of gates at any level, assuming that the gates are arranged in levels. But, to keep things simple, let's use the gate count as our main cost measure.

The number of gates depends on what the elementary operations are. We'll take these to be AND and NOT gates (and let's not count the fan-out operations). What's important is that a gate does a constant amount of computational work. If we allowed arbitrary gates then any circuit could be reduced to just one supergate, but the cost of that one gate would not be very meaningful, but since we expect large gates to be expensive to implement.

### B.3.1 Multiplication problem and factoring problem

Let's consider the *multiplication problem* where one is given two n-bit binary integers as input and the output is their product (a 2n-bit integer). For example, for inputs 101 and 111, the output should be 100011 (because the product of 5 and 7 is 35). Think of the number of bits n as being quite large, larger than the number of bits of the arithmetic logic unit of your computer. In principle, you could multiply two numbers consisting of thousands of digits by pencil and paper using the method taught in school. And it can be coded up as an algorithm (commonly referred to as the grade school multiplication algorithm). How efficient is this algorithm? assuming

you learned the algorithm that I did, its running time scales quadratically in its input size. By quadratic, I mean some constant times  $n^2$  (for sufficiently large n).

The constant depends on the exact gate set that you use and we'll often use the big-oh notation to suppress that.

**Definition B.3.1.** For  $f, g : \mathbb{N} \rightarrow \mathbb{N}$ ,  $g(n) \in O(g(n))$  if  $f(n) \leq cg(n)$  for some constant  $c > 0$  (and for sufficiently large n).

Of course if we want to implement an algorithm then we do care about what the constant multiple is. But if we're looking at things theoretically then the big-oh notation helps reduce clutter and focus attention on asymptotic growth rates, which tend to be more important for large n.

There are more efficient algorithms than the grade school method. The current record is  $O(n \log n)$  which is quite good. (There's a rich history of multiplication algorithms that evolved from  $O(n^2)$  to  $O(n \log n)$ , but we won't get into that here).

Now let's look at the factoring problem, which can be roughly thought of as the inverse of the multiplication problem. The input is an n-bit number and the output is its prime factors. For example, for input 100011, the outputs are 101 and 111 (the prime factors of 35).

You probably also learned this algorithm in grade school. For factoring numbers: first check if the number is divisible by 2. Then check for divisibility by 3. You can skip 4, since that's a multiple of 2. Then check 5, skip 6, check 7, and so on. You can stop once you get to the square root of the number because if you haven't found a divisor by then, the number must be prime.

How expensive is this computationally? For large n, there are lots of divisors to check, exponentially many. The cost is exponential in n, namely  $O(\exp(n))$ . There are more sophisticated algorithms than this method. The best currently-known classical algorithm for factoring is the so-called *number field sieve algorithm*, which scales exponentially in the cube root of n (to be more precise,  $\exp(n^{1/3} \log^{2/3}(n))$ ). Since the cube root is in the exponent, this is still essentially exponential. There is no currently-known classical algorithm for factoring whose cost scales polynomially with respect to n.

In fact, the presumed difficulty of factoring is the basis of the security of many crypto-systems.

## B.4 Classical Circuits

In classical complexity theory, a Boolean circuit is a finite directed acyclic graph with AND, OR, and NOT gates. It has n input nodes, which contain the n input

bits ( $n \geq 0$ ). The internal nodes are AND, OR and NOT gates according to the circuit, and there are one or more designated output nodes. The initial input bits are fed into AND, OR and NOT gates according to the circuit, and eventually the output nodes assume some value.

**We say that a circuit computes a Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  if the output nodes get the right value  $f(x)$  for every input  $x \in \{0, 1\}^n$ .**

A circuit family is a set  $C = \{C_n\}$  of circuits, one for each input size  $n$ . Each circuit has one output bit. Such a family recognizes or decides a language  $L \subseteq \{0, 1\}^* = \bigcup_{n \geq 0} \{0, 1\}^n$  if, for every  $n$  and every input  $x \in \{0, 1\}^n$ , the circuit  $C_n$  outputs 1 if  $x \in L$  and outputs 0 otherwise. We can think of language  $L$  as a sequence of Boolean functions  $f_n : \{0, 1\}^n \rightarrow \{0, 1\}$ , where  $f_n$  takes value 1 exactly on the  $n$ -bit strings that are in  $L$ . The circuit  $C_n$  then computes the function  $f_n$ . Such a circuit family is uniformly polynomial if there is a deterministic Turing machine that outputs  $C_n$  given  $n$  as input, using space logarithmic in  $n$ . Logarithmic space implies time that's at most polynomial in  $n$ , because such a machine will have only  $\text{poly}(n)$  different internal states, so it either halts after  $\text{poly}(n)$  steps or cycles forever. Note that the size (number of gates) of the circuit  $C_n$  can then grow at most polynomially with  $n$ . It is known that uniformly polynomial circuit family iff  $L \in P$ , where  $P$  is the class of languages decidable by polynomial-time Turing machines.

Similarly, we can consider randomized circuits. These receive, in addition to the  $n$  input bits, also some random bits ("coin flips") as input. A randomized circuit computes a function  $f$  if it successfully computes the right answer  $f(x)$  with probability at least  $2/3$  for every  $x$  (probability taken over the values of the random bits). Randomized circuits are equal in power to randomized Turing machines: a language  $L$  can be decided by a uniformly polynomial randomized circuit family iff  $L \in \mathbf{BPP}$ , where **BPP** ("Bounded-error Probabilistic Polynomial time") is the class of languages that can efficiently be recognized by randomized Turing machines with success probability at least  $2/3$ . Because we can efficiently reduce the error probability of randomized algorithms, the particular value  $2/3$  doesn't really matter here and may be replaced with any fixed constant in  $(1/2, 1)$ .

# Appendix C

## Qiskit

At the time of writing this we are using Qiskit 1.x to develop/ The following content has been copied directly from Qiskit IBM documentation.

### C.1 Introduction to Qiskit

The name "Qiskit" is a general term referring to a collection of software for executing programs on quantum computers. Most notable among these software tools is the **open-source Qiskit SDK**, and the **runtime environment** (accessed using Qiskit Runtime) through which you can execute workloads on IBM quantum computers. As quantum technology evolves, so does Qiskit, with new capabilities released every year that expands this core collection of quantum software.

In addition, many open-source projects are part of the broader Qiskit ecosystem. These software tools are not part of Qiskit itself, but rather interface with Qiskit and can provide valuable additional functionality. Parts of Qiskit environment are as shown in the figure C.1.

#### C.1.1 The Qiskit SDK

The Qisit SDK (package name qiskit) is an open-source SDK for working with quantum computers at the level of extended (static, dynamic, and scheduled) quantum circuits, operators and primitives. This library is the core component of Qiskit; it is the largest package under the Qiskit name with the broadest suit of tools for quantum computation, and many other components interface with it. Some of the most useful features of the Qiskit SDK include:

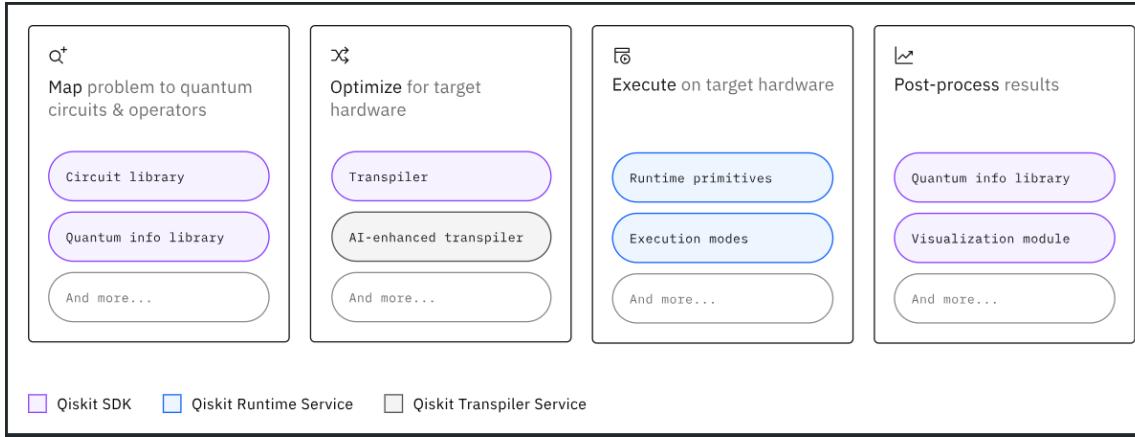


Figure C.1: Qiskit Steps

- Circuit-building tools (`qiskit.circuit`) - For initializing and manipulating registers, circuits, instructions, gates, parameters, and control flow objects.
- Circuit library (`qiskit.circuit.library`) - A vast range of circuits, instructions, and gates - key building blocks for circuit-based quantum computations.
- Quantum info library - (`qiskit.quantum_info`) - A toolkit for working with quantum states, operators and channels, using exact calculations (no sampling noise). Use this module to specify input observables and analyse fidelity of outputs from primitive queries.
- Transpiler (`qiskit.transpiler`) - For transforming and adapting quantum circuits to suit specific device topology, and optimizing for execution on real quantum systems.
- Primitives (`qiskit.primitives`) - The module that contains the base definitions and reference implementations of the **Sampler** and **Estimator** primitives, from which different quantum hardware providers can derive their own implementations. See more information about the Qiskit runtime primitives in the documentation.

## C.1.2 Qiskit Runtime

Qiskit Runtime is a cloud-based service for executing quantum computations on IBM Quantum hardware. The `qiskit-ibm-runtime` package is a client for that service, and it provides an interface to interact with the runtime service.

is the successor of the Qiskit IBM provider. The Qiskit Runtime service streamlines quantum computations and provides optimal implementation of the Qiskit primitives for IBM Quantum hardware. To get started with Qiskit runtime primitives, visit the documentation.

With Qiskit Runtime you can choose to run your quantum programs on IBM Quantum hardware through the IBM Quantum Platform or IBM Cloud. See more information on selecting an IBM Quantum channel in the documentation.

Qiskit Runtime is designed to use additional classical and quantum computer resources, including techniques such as error suppression and error mitigation, to return a higher-quality result from executing quantum circuits on quantum processors. Examples include dynamical decoupling for error suppression, and readout mitigation and zero-noise extrapolation (ZNE) for error mitigation. Learn how to configure these options on the Configure error mitigation page.

Qiskit Runtime also includes three types of execution modes for running your quantum program on IBM hardware: Job, Session, and Batch, each of which have different use cases and implications for the quantum job queue. A Job is single query to a primitive that can be run over a specified number of shots. Sessions allow you to efficiently run multiple jobs in iterative workloads on quantum computers. Batch mode allows you to submit all your jobs at once for parallel processing.

### Is Qiskit Runtime open-source?

The short answer is, not all of it. The Qiskit Runtime service software that handles the technicalities of running your quantum program on IBM Quantum device (including any error mitigation and suppression) is not open-source. However, the Qiskit Runtime client (the interface for users to access the Qiskit Runtime service), the Qiskit SDK running on the server side, and some of the software used for error mitigation, are open-source. To get involved with Qiskit open-source efforts, visit the GitHub organization at [github.com/Qiskit](https://github.com/Qiskit) and [github.com/Qiskit-Extensions](https://github.com/Qiskit-Extensions).

### C.1.3 Qiskit Serverless

Creating utility-scale quantum applications generally requires a variety of computer resource requirements. Premium users can use Qiskit Serverless (package name `qiskit-serverless`) to easily submit quantum workflows for remote, managed execution. See more information here about how to use this collection of tools.

### C.1.4 Qiskit Transpiler as a Service

The Qiskit transpiler service (package name `qiskit-transpiler` service) is a new experimental service that provides remote transpilation capabilities on the cloud to IBM Quantum Premium Plan users. In addition to the local Qiskit SDK transpiler capabilities, your transpilation tasks can benefit from both IBM Quantum cloud resources and AI-powered transpiler passes using this service. To learn more about how to integrate cloud-based transpilation into your Qiskit workflow you can check out the documentation.

### C.1.5 The Qiskit ecosystem

Beyond Qiskit there are many open-source projects that use the "Qiskit" name but are not part of Qiskit itself; rather, they interface with Qiskit and can provide valuable additional functionality to supplement the core Qiskit workflow. Some of these projects are maintained by IBM Quantum teams, whereas others are supported by the broader open-source community. The Qiskit SDK is designed in a modular, extensible way to make it easy for developers to create projects like these that extend its capabilities. Some popular projects in the Qiskit ecosystem include:

- Qiskit (`qiskit-aer`) - a package for quantum computing simulators with realistic noise models. It provides interfaces to run quantum circuits with or without noise using multiple different simulation methods. Maintained by IBM Quantum.
- qBraid SDK (`qbraid`) - a platform-agnostic quantum runtime framework for both quantum software and hardware providers, designed to streamline the full lifecycle management of quantum jobs- from defining program specifications to job submission through to the post-prcoessing and visualization of results. Maintained by qBraid.
- mthree (`mthree`) - a package for implementing M3(Matrix-free Measurement Mitigation), a measurement mitigation technique that solves for corrected measurement probabilities using a dimensionality reduction step followed by either direct LU factorization or a preconditioned iterative method that nominally converges  $\mathcal{O}(1)$  steps, and can be computed in parallel. Maintained by IBM Quantum.

You can find a catalog of projects in the Qiskit ecosystem page, as well as information about how to nominate your own project.

Please refer to the YouTube lectures by Derek Wang, research scientist at IBM for Coding with Qiskit 1.x. Qiskit version 1.x for utility scale for Quantum Computing. From installing Qiskit version1.x from scratch to learning how to run Quantum circuits both Unitary and dynamic based on some of the recent research papers by IBM Quantum. Using devices over 100 qubits using latest air suppression and mitigation techniques. Then learning how to contribute to the Qiskit ecosystem through open-source extra ordinate Abby Mitchell.

This is to get you up to speed with the latest developments on quantum computers for your own work. The foundation and focused on Qiskit 1 - an open source and freely available software development kit that allows you to program useful Quantum computational workflows. From designing Quantum Circuits and designing Quantum algorithms to submitting them to real Quantum Computers and orchestrating large scale complex workloads. Qiskit has now been for almost six years with previous iterations with Coding with Qiskit series. In the past few years with major developments in Quantum Computing most notably with the onset of the Utility era where Quantum Computers have now reached the level of scale and reliability such that researchers can now use them as legitimate tool for scientific exploration as highlighted by Research Nature paper by IBM Quantum.

In utility era, the Quantum computational workflows will grow only in scale and complexity. There are high immpact scientific papers that already use more than 100 qubits on IBM's 127 qubits chips, with thousands of gates in the circuit and tens and thousands of circuits in a workflow. With the proliferation of methods like circuit cutting and error mitigation that require fluid orchestration between multiple classical and quantum computers. The technology has evolved dramatically. Qiskit 1.0 is highly performant - faster and takes up less memory making it convenient for faster transpiler circuits with 100 plus qubits and thousands of gates. It is learner, removed meta package architecture, spun off packages like qiskit.algorithms into a seperate one and deprecated modules like qiskit.opflow to make it more focused and maintainable. It is more stable. More features like Primitives in the form of sampler and estimator are now the access model for qunautm hardware. Improved upon backend.run method from pre 1.0 versions. Primitives ensure that access to quantum hardware is consistent and reliable. Qiskit 1.0 is ready for dynamic circuits - enabling complex logic within a Quantum circuit more efficiently generating entangled states, teleport long range gates and path toward error correction. It is now easier to build on Qiskit. Like build your own transpiler plugin. extend qiskit capabilities for your own needs and share your code throughout the Qiskit ecosystem.

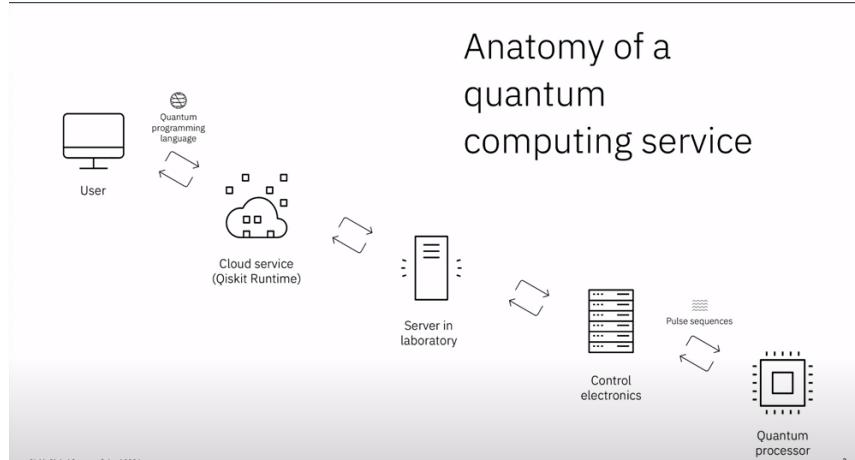


Figure C.2: Anatomy of Quantum Computing Service

## C.2 Install Qiskit

Whether you will work locally or in a cloud environment, the first step for all users is to install Qiskit. For those wanting to run on a real system, your next step is to choose one of two channels in order to access IBM Quantum systems: IBM Quantum Platform or IBM Cloud.

Checkout the IBM Quantum Qiskit guide on how to install Qiskit and Qiskit Runtime and setup an IBM Quantum channel.

## C.3 Anatomy of a Quantum Computing Service

How does a Quantum program go from being on your laptop with you as the end user through to actually being run on a real quantum device? You need to use some Quantum programming language and in this case its going to be Qiskit to construct a quantum program. Then using a Cloud based service in this case called Qiskit Runtime to package up that program and send it through the cloud until it hits a server in a laboratory somewhere. Then the program will go through control electronics. The outcome of this will be your program will be converted to microwave pulse sequences that can then be run on real quantum processor. Once your program is run we will get the results back which will get propagated all the way back up the chain, back to you, the end user on your laptop. This gives you an overview at the high level view of the process and indicates the stuff going behind the scenes that

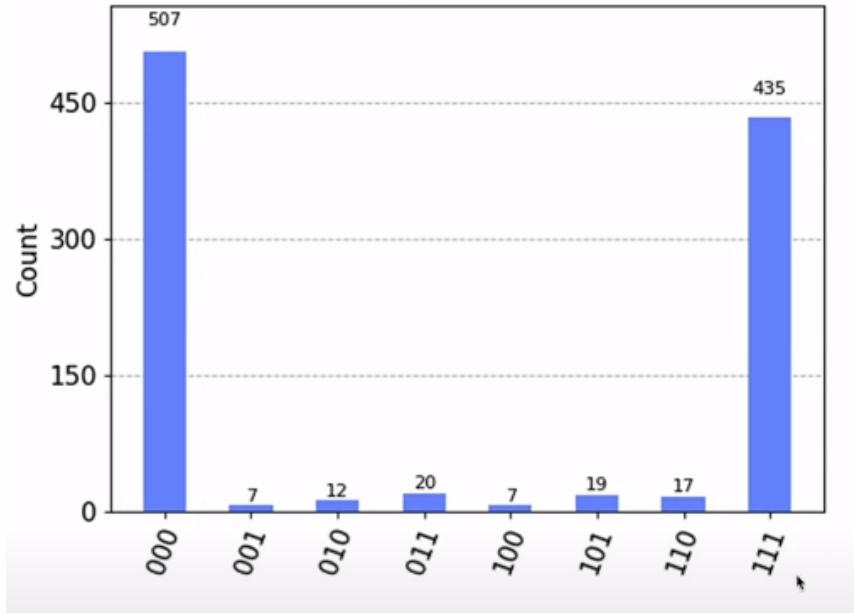


Figure C.3: Example of final result

maybe you as the end user aren't aware of. For the purpose of this lecture we are going to be focusing on user and Qiskit runtime (the parts with which user will be directly interacting with).

An example of final result is as shown in the figure C.3. You will get a series of bit strings many zeros and ones, these in a classical format. You will typically have run the program many times on the quantum device and so you will get probabilities back on how did you get a certain bit string more so than others. Thus as can be seen through the graph that the quantum program gave a lot of results that were 000 and 111 which is probably the answer to the program that you are trying to run. We also get a lot of much smaller counts of different bitstrings here. And this, shows that there are some errors on the quantum device. Quantum devices are not perfect. You will often get small numbers of incorrect bit strings from the outcome of your experiment. This is why you have to run your program multiple times so that you can start to see these are where the errors are. We can also talk about ways to prevent or mitigate these kinds of errors. Ideally we would only want to see two bars in the image.

**What is Qiskit?**

- Qiskit SDK is an open-source SDK for working with Quantum Computers at the level of extended quantum circuits, operators and primitives. You can find the code and documentation here. Currently version 1.0 is used released on Feb 2024.
- Qiskit runtime is a cloud based service for executing quantum computations on IBM Quantum hardware. Qiskit runtime services is not open-source but the client that you use to interact with that service is. And you can check that repository on GitHub along with the documentation. Currently version 2 is used (released March 2024).
- Qiskit patterns is a framework for breaking down domain-specific problem into stages to build a quantum program. It is guidelines on how to construct a full quantum program including everything from building your initial circuit through to running it on real hardware and getting the results at the end.
- The Qiskit Ecosystem is a collection of software and projects that build or extend Qiskit. These are additional packages which are building on Qiskit which can be an addition to your toolbox of Qiskit SDK and Qiskit runtime. These are referred to as part of the Qiskit ecosystem. It can be found here to find the catalog some maintained by IBM and some maintained by members of the open source community.

### C.3.1 Getting set up with Qiskit

```
1 pip install 'qiskit[visualization]' # Install Qiskit SDK with the  
    additional visualization capabilities  
2 pip install qiskit-ibm-runtime
```

For visualizing circuits and results install Qiskit with visualization add-on.

# Bibliography

- [1] Ronald de Wolf. Quantum computing: Lecture notes, 2023.
- [2] Lin Lin. Lecture notes on quantum algorithms for scientific computation, 2022.
- [3] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*, volume 2. Cambridge university press Cambridge, 2001.
- [4] Ramamurti Shankar. *Principles of quantum mechanics*. Springer Science & Business Media, 2012.