

Quantum Computing

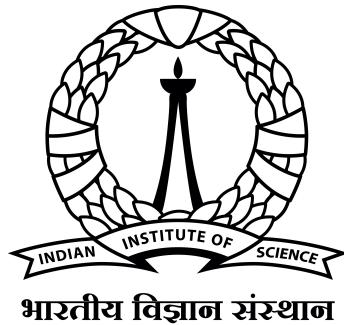
Nihar Shah

January 30, 2025

Quantum Computing and Quantum Information: Theory and Practice

Nihar Shah

Guide: Professor Phani Motamarri



*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, Bengaluru. The work is done under the guidance of Prof. Phani Motamarri. This work includes notes made during the 1 year work from June 2024 to August 2025 and is 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 starting with the Appendix to get a good grasp of the math used in the entire text. The work is divided into parts: The first part is the introduction to Quantum Computing, the second part is Quantum Linear Algebra, the third part is on Quantum Information Theory and at last the fourth part is on Quantum Error Correction. 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 Motamarri, 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

Acknowledgements	ii
I Quantum Computing	1
1 Introduction	2
1.1 Why Quantum Computation?	2
1.2 The Double Slit Experiment	4
1.3 The Coin Experiment: Analogy for intuition	8
1.4 Quantum Information	11
1.4.1 The Classical Information	11
1.4.2 A probabilistic model	12
1.4.3 Quantum bits (qubits)	14
1.4.4 Example of Qubits	20
1.4.5 Multiple Qubits	21
1.5 Measurements	33
1.5.1 Simple Measurements	34
1.5.2 Measuring Multiple Systems	41
1.6 The Stern-Gerlach Experiment	45
1.7 Qiskit Examples	49
1.7.1 Vector and Matrices in Python	49
1.7.2 States, measurements and operations	50
1.7.3 Partial measurements	57
2 Axioms of Quantum Mechanics	59
2.1 Postulate 1: The State Space Postulate	59
2.2 Postulate 2: Evolution Postulate	61
2.3 Postulate 3: Measurement Postulate	64
2.3.1 Projective Measurements	68

2.3.2	Positive Operator Valued Measure (POVM) measurements	70
2.4	Postulate 4: Composite Systems	74
2.4.1	Full and Partial Measurements of a multi-particle systems	76
3	Qubits and quantum gates	80
3.1	Phase	81
3.2	Bloch-Poincare sphere representation of a Qubit	82
3.3	Single Qubit Gates	87
3.3.1	Pauli Matrices	88
3.3.2	Hadamard Gate	99
3.3.3	S and S^\dagger Gate	102
3.3.4	T and T^\dagger Gate	105
3.3.5	Quantum Rotation Gates	111
3.3.6	Universal Quantum Gates	114
3.4	Multi-Qubit Gates	124
3.4.1	Two-qubit Gates	125
3.4.2	Three-qubit Gates	139
3.5	Implementation of Controlled-U	146
3.6	Universal quantum gates	151
3.6.1	Two-level unitary gates are universal	153
3.6.2	Single qubit and CNOT gates are universal	156
3.6.3	Approximating arbitrary unitary gates	160
3.7	Measurements	162
3.7.1	Principle of deferred measurement	162
3.7.2	Principle of implicit measurement	163
4	Quantum Circuits	166
4.1	Classical Circuits simulating Quantum Circuits	175
4.2	Universal gate sets	177
4.2.1	A discrete set of universal operations	178
4.2.2	Linear Growth and Duhamel's Principle	180
4.3	Reversible Computation	183
4.3.1	Quantum Circuits simulating Classical Circuits	183
4.3.2	Fixed Point Number Representation and Classical Arithmetic operations	186
4.4	Energy and computation	187
4.4.1	The billiard ball model of computation	189

4.4.2	From Reversible classical computations to quantum computations	199
4.4.3	reversible and Quantum versions of simple classical gates	200
4.5	Reversible implementation of Classical Circuits	202
4.5.1	A Naive reversible Implementation	203
4.5.2	A General Construction	204
4.5.3	Reversible Computation	205
4.5.4	Reversible Implementation of invertible functions	210
4.6	Complexity of Quantum Algorithms	211
4.7	Oracles	213
4.8	Quantum Algorithms as Quantum Circuits	218
4.9	Fault tolerant Computation	220
5	Number-Theoretic Problems	221
5.1	Arithmetic Problems	221
5.1.1	Integer Addition	221
5.1.2	Integer Multiplication	222
5.1.3	Integer Division	222
5.1.4	Greatest Common Divisor	223
5.1.5	Modular Integer Addition	223
5.1.6	Modular integer Multiplication	223
5.1.7	Modular Inverse	223
5.1.8	Modular Exponentiation	223
5.1.9	Primality testing	223
5.1.10	Integer Factoring	224
6	Entanglement	225
6.1	Two qubits	225
6.2	Bell States/EPR Pairs (EPR Paradox)	228
6.3	Bell's Thought Experiment	230
6.4	No Cloning Theorem	235
6.5	Applications of Entanglement	237
6.5.1	The CHSH Game	237
6.5.2	Quantum Teleportation	242
6.5.3	Superdense Coding	249
7	Elitzur-Vaidman Bomb Detection Algorithm	261
7.1	The Quantum Zeno Effect	261
7.2	The Problem	262

7.2.1	Approach 1: Naive Approach	262
7.2.2	Approach 2: Quantum Superposition Approach	262
7.2.3	Elitzur-Vaidman Bomb Detection Algorithm	264
8	Deutsch's Algorithm	267
8.1	The Problem Definition	267
8.2	The Classical Solution	268
8.3	The Quantum Solution: Deutsch's Algorithm	269
8.3.1	Quantum Parallelism	269
8.3.2	Idea of the Algorithm	272
8.3.3	Algorithm	272
8.3.4	Example	276
8.4	Implementation using Qiskit	279
8.4.1	Background	279
8.4.2	Requirements	279
8.4.3	Setup	279
9	Deutsch-Jozsa Algorithm	285
9.1	The Problem Definition	285
9.2	The Classical Solution	285
9.3	The Quantum Solution: Deutsch-Jozsa Algorithm	287
9.3.1	Idea of the Algorithm	287
9.3.2	Algorithm	287
9.3.3	Example	291
9.4	Complexity Analysis	294
9.5	Conclusion	295
9.6	Implementation using Qiskit	295
9.6.1	Background	295
9.6.2	Requirements	295
9.6.3	Setup	297
10	Bernstein-Vazirani Algorithm	307
10.1	The Problem	307
10.2	The Classical solution	308
10.3	Quantum solution	308
10.4	Implementation	310

11 Simon's Algorithm	311
11.1 The Problem Definition	311
11.2 The Classical Solution	313
11.2.1 Classical Lower Bound	314
11.3 The Quantum Solution: Simon's Algorithm	316
11.3.1 Algorithm	317
11.3.2 Example	322
11.4 Complexity Analysis	325
11.5 Conclusion	325
12 Discrete log Problem	327
12.1 Definition of \mathbb{Z}_m and \mathbb{Z}_m^*	327
12.2 Generators of \mathbb{Z}_p^* and the exponential/log functions	328
12.3 Discrete exponential problem	329
12.3.1 Repeated Squaring Trick	329
12.4 Discrete log problem	330
12.5 Shor's Algorithm for the discrete log problem	330
12.5.1 Shor's function with a property similar to Simon's	330
13 Quantum Fourier Transform	333
13.1 Roots of Unity	334
13.2 Classical Fast Fourier Transform	335
13.3 Discrete Fourier Transform Matrix	336
13.3.1 Properties of QFT	338
13.4 Quantum Fourier Transform with quantum gates	345
13.5 Efficient implementation of the Quantum Fourier Transform	347
13.6 Summary	351
13.7 Analysis	358
13.7.1 Total Gate Complexity	358
13.7.2 Comparison with Discrete Fourier Transform	359
13.8 Implementation using Qiskit	359
13.8.1 Quantum Fourier Transform	359
13.8.2 Inverse Quantum Fourier Transform	362
14 Phase Estimation	365
14.1 Phase Estimation	365
14.2 The Problem Definition	366
14.3 The Classical Solution	367
14.4 The Quantum Solution	367

14.4.1 Hadamard Test	367
14.4.2 Overlap Estimate using Swap Test	370
14.4.3 Overlap estimate with Relative phase information	371
14.4.4 Single Qubit Phase Estimation	372
14.5 Kitaev's Method - for Quantum Phase Estimation	375
14.5.1 Idea of the Algorithm	375
14.5.2 Algorithm	377
14.5.3 Example	379
14.5.4 Validity of Kitaev's Algorithm	379
14.6 Quantum Phase Estimation using Quantum Fourier Transform	380
14.6.1 Implementation of Controlled Unitary operation	382
14.6.2 Implementation of QPE Circuit	385
14.7 Special Case: Superposition of Eigenvectors	396
14.8 Complexity Analysis	398
14.9 Implementation using Qiskit	402
15 Order Finding	403
15.1 Introduction	403
15.2 The Problem Definition	406
15.3 The Classical Solution	406
15.4 The Quantum Solution	408
15.4.1 Multiplicity-controlled $-U_{a,N}$ gate	411
15.4.2 Precision needed to determine $1/r$	415
15.4.3 Can we construct $ \psi_1\rangle$?	416
15.4.4 Order-finding using a random eigenvector $ \psi_k\rangle$	417
15.4.5 Conclusions of order-finding with a random eigenvector	423
15.4.6 Order-finding using a superposition of eigenvectors	423
15.5 Performance	426
15.6 Complexity Analysis	428
15.7 Summary	429
16 Shor's Algorithm	430
16.1 Introduction	430
16.2 Problem Definition	431
16.3 Reduction of factoring to order-finding	432
16.4 Example	434
16.5 Summary	435

17 Period-finding	437
17.1 Problem Definition	437
18 Hidden Subgroup problem	439
19 Grover's Search Algorithm	440
19.1 The Problem Definition	441
19.2 The Classical Solution	441
19.3 The Quantum Solution: Grover's Algorithm	443
19.3.1 Idea of the Algorithm	443
19.3.2 Algorithm	447
19.3.3 Example	463
19.4 Complexity Analysis	469
19.4.1 Unique Search Problem	470
19.4.2 Multiple Solutions	472
19.4.3 Trivial Case	476
19.5 Optimality of Grover's algorithm	477
19.6 Conclusion	484
19.7 Quantum Counting	485
19.8 Amplitude Estimation	488
19.9 Amplitude Amplification	490
19.10 Applications of Grover's Algorithm	492
19.10.1 Application: Satisfiability	492
19.10.2 Speeding up the solutions of NP-complete problems	493
19.10.3 Quantum Search of an unstructured database	495
19.11 Implementation using Qiskit	498
19.11.1 Background	498
19.11.2 Requirements	498
19.11.3 Setup	499
20 Trotter based Hamiltonian Simulation	506
20.1 Simulation of Quantum Systems	507
20.1.1 Simulation in action	508
20.2 Introduction	510
20.3 Hamiltonian Simulation	516
20.3.1 Evolution of a Closed Quantum System	518
20.4 The Problem Definition	518
20.5 Trotter Splitting	519

20.6 Evolution of Hamiltonian formed by Pauli Operators: Quantum circuits and exact simulation	527
20.6.1 One-qubit Hamiltonians	527
20.6.2 Two-qubit Hamiltonians	527
20.7 How to simulate any Hamiltonian	529
20.8 Example	530
20.9 Commutator type error bound	533
21 The HHL Algorithm	536
21.1 Introduction	536
21.2 The Linear-System Problem (LSP)	536
21.3 The HHL Algorithm	538
21.3.1 Some Mathematical Preliminaries	538
21.3.2 Algorithm	540
21.3.3 Example	547
21.4 Complexity Analysis	551
21.5 Improvements in HHL algorithm	555
21.6 Implementation using Qiskit	557
21.7 Applications	557
21.7.1 Poisson's problem	557
21.7.2 Solve Linear Differential Equations	560
22 Quantum Walk Algorithms	566
22.1 Classical Random Walks	566
22.2 Quantum walks	568
22.2.1 Continuous-time quantum walk	571
23 Variational Algorithms	572
23.1 Variational Theorem	575
23.1.1 Mathematical intuition for energy and ground state	575
23.1.2 Variational theorem of Quantum Mechanics	576
23.2 Reference States	577
23.2.1 Default State	577
23.2.2 Classical Reference state	577
23.2.3 Quantum Reference State	578
23.2.4 Constructing Reference States using template circuits	578
23.2.5 Application Specific Reference States	579
23.3 Ansätze and Variational Forms	580
23.3.1 Parameterized Quantum Circuits	580

23.3.2 Variational Form and Ansatz	582
23.3.3 Problem-specific ansatze	586
23.4 Cost Functions	589
23.4.1 Primitive	589
23.4.2 Cost functions	598
23.4.3 Summary	615
23.5 Optimization Loops	616
23.5.1 Local and Global Optimizers	616
23.5.2 Gradient-Based and Gradient-Free Optimizers	619
23.5.3 Barren Plateaus	621
23.5.4 Summary	623
23.6 Instances and Extensions	624
23.6.1 Variational Quantum eigensolver (VQE)	624
23.6.2 Subspace Search VQE (SSVQE)	629
23.6.3 Variational Quantum Deflation	632
23.6.4 Quantum Sample Regression (QSR)	639
23.6.5 Summary	641
23.7 Variational Quantum Eigenlsolver	642
23.7.1 Background	642
23.7.2 Requirements	642
23.7.3 Setup	643
II Quantum Linear Algebra	649
24 Block Encoding	651
24.1 Motivation	651
24.2 Introduction	652
24.3 Query model for matrix entries	654
24.4 Block Encoding	655
24.4.1 The Definition	660
24.5 Unitary Matrices - Trivial Block Encoding	664
24.6 Block encoding of density operators	664
24.7 Block encoding of POVM operators	665
24.8 Block-encoding of Gram matrices	666
24.9 Block Encoding of s-sparse matrix	667
24.9.1 Banded Circulant Matrix	676
24.10 Hermitian Block Encoding	677

24.11	Query models for general sparse matrices	678
24.12	Resource cost -gates and qubits	681
24.13	Example Use Cases	683
25	Linear Combination of Unitaries	685
25.1	Introduction	685
25.2	Manipulating block-encodings	689
25.2.1	Products	690
25.2.2	Tensor products	692
25.2.3	Linear Combinations	692
25.3	Examples	694
26	Matrix functions of Hermitian Matrices	698
27	Qubitization	700
27.0.1	Motivation	700
27.0.2	Overview	701
27.1	Qubitization of Hermitian matrices with Hermitian block encoding . .	702
27.2	Qubitization of hermitian matrices with general block encoding . . .	705
27.3	Dominant resource cost - qubits and gates	707
27.4	Example - Use cases	709
28	Quantum Eigenvalue Transformation	710
28.1	Hermitian Block encoding	710
28.1.1	General Block Encoding	714
28.1.2	General Matrix Polynomials	715
29	Quantum Signal Processing	716
29.1	Introduction	716
29.1.1	Overview	716
29.2	QSP for real polynomials	721
29.3	Dominant resource cost - gates and qubits	722
30	Quantum Singular Value Transformation	725
30.1	Generalized Matrix functions	725

III Quantum Complexity Theory	727
31 Computational Complexity Theory	728
31.1 Introduction	728
31.1.1 Efficient vs Inefficient	729
31.1.2 The Challenges	730
31.1.3 What other problems?	732
31.2 The power of quantum computation	733
31.2.1 Why study classical computer science?	734
31.2.2 Computational Complexity Theory	735
31.3 Models of Computation	737
31.3.1 Turing Machines	738
32 Loading classical data	739
32.1 Quantum random access memory	739
32.1.1 Resource cost	740
IV Quantum Information Theory	741
33 Introduction to Quantum Information	742
33.1 Density Matrices	743
33.2 Operations on Density matrices	747
33.3 Postulates	751
33.3.1 Postulate 1: The State Postulate	751
33.3.2 Postulate 2: Evolution Postulate	751
33.3.3 Postulate 3: Measurement Postulate	751
33.3.4 Postulate 4: Composite Systems	753
33.4 The Reduced Density Operator	757
V Quantum Error Correction	760
34 Classical Error Correction	761
34.1 Background and Motivation	761
34.2 Error-Detecting Code	764
34.2.1 Memoryless Binary symmetric channels	766
34.2.2 Classical repetition codes	766
34.3 Error-Correcting Code	768

34.3.1	The Length-7 Hamming Code	768
34.4	Linear Codes	771
34.4.1	Minimum Distance	773
34.4.2	Error Correction	774
34.4.3	Erasures	774
35	Quantum Noise and Quantum Error-Correcting Codes	776
35.1	Repetition code for qubits	777
35.1.1	Encoding	777
35.1.2	Bit-Flip error detection	778
35.2	Phase-flip errors	779
35.2.1	Modified repetition code for phase-flip errors	780
35.3	The 9-qubit Shor code	781
35.3.1	Code description	782
A	Linear Algebra and Calculus Prerequisites	787
A.1	Linear Algebra	788
A.1.1	Mathematical Notation	788
A.1.2	Basics of Linear Algebra	788
A.1.3	Inner Product	795
A.1.4	Gram-Schmidt Orthogonalization	801
A.1.5	Cauchy-Schwarz and Traingle Inequalities	802
A.1.6	Linear Operators and Matrix Representation	803
A.1.7	Hermitian and Unitary Operators	813
A.1.8	Positive Operators	819
A.1.9	Eigen Value Problems	820
A.1.10	Spectral Decomposition	828
A.1.11	Commutator and Anti-Commutator	829
A.1.12	Simultaneous Diagonalization of two Hermitian Operators	830
A.1.13	Operator Functions	831
A.1.14	Trace of a Matrix	836
A.1.15	Norms	838
A.1.16	Induce Matrix Norm	839
A.1.17	Conditioning and Condition Number	840
A.2	Tensor Products	841
A.3	Complex Numbers	848
A.3.1	Complex Plane	848
A.3.2	Addition of Complex Numbers	849

A.3.3	Multiplication of Complex Numbers	849
A.3.4	Complex Conjugate	850
A.3.5	Modulus of a Complex Number	850
A.3.6	Polar form	851
A.3.7	Euler's Formula	851
A.4	Probability Theory	853
A.5	Calculus	853
A.6	Group Theory	853
A.6.1	Basic Definitions	853
A.6.2	Generators	855
A.7	Number Theory	856
A.7.1	Fundamentals	856
A.7.2	Modular Arithmetic and Euclid's Algorithm	858
A.7.3	Advanced Properties:	860
A.8	Reduction of factoring to order-finding	871
A.9	Continued fractions	872
B	Classical Computations	876
B.1	Classical Logic Gates	877
B.1.1	NOT Gate	877
B.1.2	AND Gate	877
B.1.3	OR Gate	878
B.1.4	NAND Gate	878
B.1.5	NOR Gate	879
B.1.6	XOR Gate	880
B.1.7	FANOUT Gate	880
B.1.8	CROSSOVER Gate	881
B.2	Classical Circuits	881
B.3	Classical Algorithms as logic circuits	883
B.3.1	Multiplication problem and factoring problem	884
B.4	Classical Circuits	885
C	Qiskit	887
C.1	Introduction to Qiskit	887
C.1.1	The Qiskit SDK	887
C.1.2	Qiskit Runtime	888
C.1.3	Qiskit Serverless	889
C.1.4	Qiskit Transpiler as a Service	890

C.1.5	The Qiskit ecosystem	890
C.2	Install Qiskit	892
C.3	Anatomy of a Quantum Computing Service	892
C.3.1	Getting set up with Qiskit	894
	Bibliography	895

Part I

Quantum Computing

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.

This chapter introduces the basic framework of quantum information, including the description of quantum states as vectors with complex number entries, measurements that allow classical information to be extracted from quantum states, and operations on quantum states that are described by unitary matrices. We will restrict our attention in this chapter to the comparatively simple setting in which a single system is considered in isolation. We will then expand our view to multiple systems, which can interact with one another and be correlated, for instance.

There are, in fact, two common mathematical descriptions of quantum information. The one introduced in this lesson is the simpler of the two. This description is sufficient for understanding many (or perhaps most) quantum algorithms, and is a natural place to start from a pedagogical viewpoint.

A more general, and ultimately more powerful description of quantum information, in which quantum states are represented by density matrices, will be introduced in a later lesson. The density matrix description is essential to the study of quantum information, for several reasons. As examples, they can be used to model the effects of noise on quantum computations, or the state of one piece of an entangled pair.

More generally, density matrices serve as a mathematical basis for quantum information theory and quantum cryptography, and are quite beautiful from a mathematical perspective. For these reasons, we encourage you to learn more about it when the time is right, but for now our focus will be on the simpler description of quantum information.

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 slit. The bullets that do go through the slits then land on the observing screen behind them. Now suppose we close 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

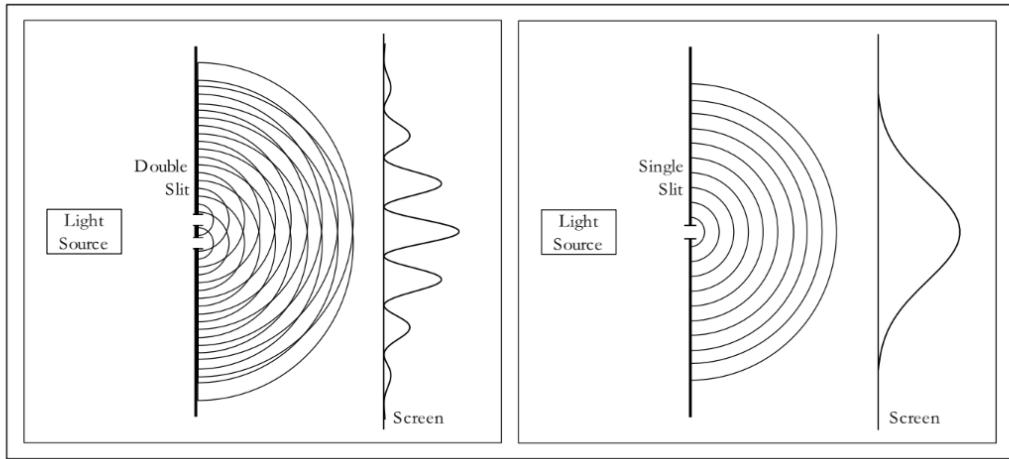


Figure 1.1: Double slit and single-slit diffraction

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 a 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

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 a 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 thought of as a paradox. The resolution 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 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 (i.e. without tossing the coin) 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.

1.4 Quantum Information

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

1.4.1 The Classical Information

To describe quantum information and how it works, we will begin with an overview of classical information. Some readers may wonder why we choose to devote so much

attention to classical information in a course on quantum information, but there are good reasons. For one, although quantum and classical information are different in some pretty spectacular ways, their mathematical descriptions are actually quite similar.

Classical information also serves as a familiar point of reference when studying quantum information, as well as a source of analogy that goes a surprisingly long way. It is common that people ask questions about quantum information that have natural classical analogs — often with simple answers that can provide both clarity and insight into the original questions about quantum information. Indeed, it is not at all unreasonable to claim that one cannot truly understand quantum information without understanding classical information.

Some readers may already be familiar with the material to be discussed in this section, while others may not — but the discussion is meant for both audiences. In addition to highlighting the aspects of classical information that are most relevant to an introduction to quantum information, this section introduces the Dirac notation, which is often used to describe vectors and matrices in quantum information and computation. As it turns out, the Dirac notation is not specific to quantum information: it can equally well be used in the context of classical information, as well as for many other settings in which vectors and matrices arise

1.4.2 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 Σ 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 Σ , and when we write such a vector in the above form we are using the most natural way of ordering the elements of Σ . 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 Σ . 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 Σ is not necessarily $\{0, 1\}$. In general the dimension of the vectors and matrices will be equal to the size of Σ .

1.4.3 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 Σ 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 Σ 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 Σ , 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 θ in the case of R_θ). 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 1.4.1. 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 1.4.1. 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 α and β means that $|\psi\rangle$ is normalized. These complex numbers α and β 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 1.4.2. 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 α_i means that $|\psi\rangle$ is a vector of length of 1.

Example 1.4.2. 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 1.4.3. Verify that for all values of θ , $|\psi\rangle = \cos(\theta)|0\rangle + \sin(\theta)|1\rangle$ 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 θ .

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 1.4.3. 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 1.4.4. 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 1.4.4. 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 1.4.5. Express $|1\rangle$ in the Hadamard basis. That is, find coefficients α and β 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 α, β 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 $|v\rangle$.

1.4.4 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 $|+\rangle$, $|-\rangle$ 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").

1.4.5 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 Σ 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 4×4 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 4×4 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 Σ , instead of Σ 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 α_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 α_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 1.4.5. 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 1.4.6. 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 1.4.6. 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 join 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 1.4.7. 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 α , 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 1.4.8. 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 1.4.7. 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 1.4.1. *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 1.4.9. 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 1.4.10. 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 1.4.11. 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 1.4.12. 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

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

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.

1.5 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.

1.5.1 Simple Measurements

This linear superposition $|\psi\rangle = \sum_{j=0}^{k-1} \alpha_j |j\rangle$ 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 α_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 α_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 ψ 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 α, β 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

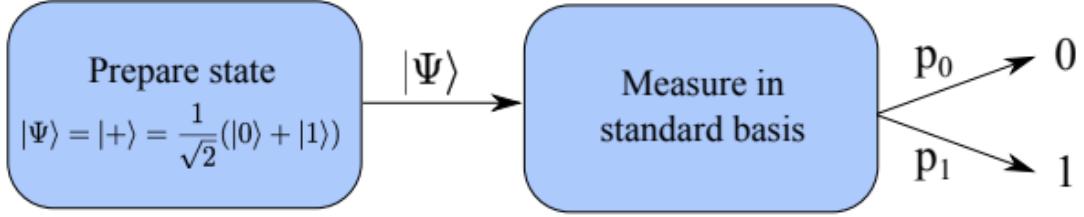


Figure 1.2: Genuine randomness from the preparation of a qubit in superposition

products

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

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

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 1.5.1. 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 1.5.2. 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 1.5.3. 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

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

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

$$\begin{aligned}p_0 &= |\langle 0|\psi\rangle|^2 = \left| (1 \ 0) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \right|^2 = |\alpha|^2 \\ p_1 &= |\langle 1|\psi\rangle|^2 = \left| (0 \ 1) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \right|^2 = |\beta|^2\end{aligned}$$

Example 1.5.4. 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 1.5.1. 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 1.5.5. 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:

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

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

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 1.5.6. 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, θ ? 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 θ . 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 ± 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.

1.5.2 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 1.5.7. 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 1.5.8. 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

$$\begin{aligned} p_{00} &= p_{11} = \frac{1}{2} \\ p_{01} &= p_{10} = 0 \end{aligned}$$

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

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

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

$$\begin{aligned} Pr[\text{outcome is } 00] &= \frac{1}{4} & Pr[\text{outcome is } 01] &= 0 \\ Pr[\text{outcome is } 10] &= \frac{1}{4} & pr[\text{outcome is } 11] &= \frac{1}{2} \end{aligned}$$

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 1.5.9. 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

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

1.6 The Stern-Gerlach Experiment

The concepts of superposition and entanglement are counter-intuitive to our daily experiences. The following experiment proves that the nature truly operates in this peculiar manner. It provides the proof for existence of qubits in Nature. thus, showing that the realization of large-scale quantum computers will be experimentally feasible.

The qubit is a fundamental element for quantum computation and quantum information. An early experiment indicating the qubit structured was performed by Stern in 1921 and Gerlach in 1922 in frankfurt. In the original Stern-Gerlach Experiment, hot atoms were 'beamed' from an oven through a magnetic field which caused the atoms to be deflected, and then the position of each atom was recorded. The original experiment was done with silver atoms, which have a complicated structure that obscures the effects we are discussing. What we describe below actually follows a 1927 experiment done using hydrogen atoms. The same basic effect is observed, but with hydrogen atoms the discussion is easier to follow. Keep in mind, though, that this privilege wasn't available to people in the early 1920s, and they have to be very ingenious to think up explanations for the more complicated effects they observed.

Hydrogen atoms contain a proton and an orbiting electron. you can think of this electron as a little 'electric current' around the proton. This electric current causes

the atom to have a magnetic field; each atom has what physicists call a ‘magnetic dipole moment’. As a result each atom behaves like a little bar magnet with an axis corresponding to the axis the electron is spinning around. throwing little bar magnet is through a magnetic field causes the magnets to be deflected by the field, and we expect to see a similar deflection of atoms in the Stern-Gerlach experiment.

How the atom is deflected depends upon both the atom’s magnetic dipole moment - the axis the electron is spinning around and the magnetic field generated by the Stern-Gerlach device. We won’t go through the details, but suffice to say that by constructing the Stern-Gerlach device appropriately, we can cause the atom to be deflected by an amount that depends upon the \hat{z} component of the atom’s magnetic dipole moment, where \hat{z} is some fixed external axis.

Two major surprises emerge when this experiment is performed. First, since the hot atoms exiting the oven would naturally be expected to have their dipoles oriented randomly in every direction, it would follow that there would be a continuous distribution of atoms seen at all angles exiting from the Stern-gerlach device. Instead, what is seen is atoms emerging from a discrete set of angles. Physicists were able to explain this by assuming that the magnetic dipole moment of atoms is quantized, that is, comes in discrete multiples of some fundamental amount.

This observation of quantization in the Stern-Gerlach experiment was surprising to physicists of the 1920s, but not completely astonishing because evidence of quantization effects in other systems was becoming widespread at that time. What was truly surprising was the number of peaks seen in the experiment. The hydrogen atoms being used were such that they should have zero magnetic dipole moment. Classically, this is surprising in itself, since it corresponds to no orbital motion of the electron, but based on what was known of quantum mechanics at that time was an acceptable notion. Since the hydrogen atoms would therefore have zero magnetic moment, it was expected that only one beam of atoms would be seen, and this beam would not be deflected by the magnetic field. instead, two beams were seen, one deflected up by the magnetic field and the other deflected down.

This puzzling doubling was explained after considerable effort by positioning that the electron in the hydrogen atoms has associated with it a quantity called spin. This spin is not in any way associated to the usual rotational motion of the electron around the proton; it is an entirely new quantity associated with an electron. The great physicist Heisenberg labeled the idea ‘brave’ at the time it was suggested, and it is a brave idea, since it introduces an essentially new physical quantity into Nature. The spin of the electron is posited to make an extra contribution to the magnetic dipole moment of a hydrogen atom, in addition to the contribution due to the rotational motion of the electron. What is the proper description of the spin

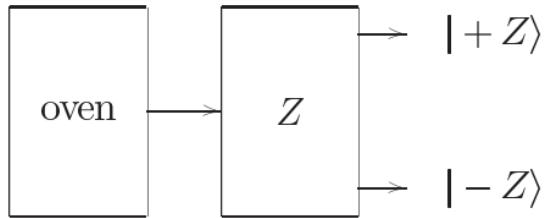


Figure 1.3: Abstract schematic of Stern-Gerlach experiment. Hot hydrogen atoms are beamed from an oven through a magnetic field, causing a deflection either up ($|+Z\rangle$) or down ($| - Z \rangle$)

of the electron? As a first guess, we might hypothesize that the spin is specified by a single bit, telling the hydrogen atom to go up or down. Additional experimental results provide further useful information to determine if this guess needs refinement or replacement. Let's represent the original Stern-Gerlach apparatus as shown in the figure 1.5. Its outputs are two beams of atoms, which we shall call $|+Z\rangle$ and $| - Z \rangle$. (We're using suggestive notation which looks quantum mechanics, but of course you're free to use whatever notation you prefer). Now suppose we cascade two Stern-Gerlach apparatus together, as shown in the figure 1.4. We arrange it so that the second apparatus is tipped sideways, so the magnetic field deflects the atoms along the \hat{x} axis. In our thought-experiment we'll block off the $| - Z \rangle$ output from the first Stern-Gerlach apparatus, while the $|+Z\rangle$ output is sent through a second apparatus oriented along the \hat{x} axis. A detector is placed at the final output to measure the distribution of atoms along the \hat{x} axis.

A classical magnetic dipole pointed in the $+ \hat{z}$ direction has no net magnetic moment in the \hat{x} direction, so we might expect that the final output would have one central peak. However, experimentally it is observed that there are two peaks of equal intensity! So perhaps these atoms are peculiar, and have definite magnetic moments along each axis, independently. That is, maybe each atom passing through the second apparatus can be described as being in a state we might write as $|+Z\rangle|+X\rangle$ or $|+Z\rangle|-X\rangle$, to indicate the two values for spin that might be observed. Another experiment, shown in figure 1.4, can test this hypothesis by sending one beam of the previous output through a second \hat{z} oriented Stern-Gerlach apparatus. If the atoms had retained their $|+Z\rangle$ orientation, then the output would be expected to have only one peak at the $|+Z\rangle$ output. However, again tow beams are observed at the final output, of equal intensity. Thus, the conclusion would seem to be that contrary to classical expectations, a $|+Z\rangle$ state consists of equal portions of $|+Z\rangle$

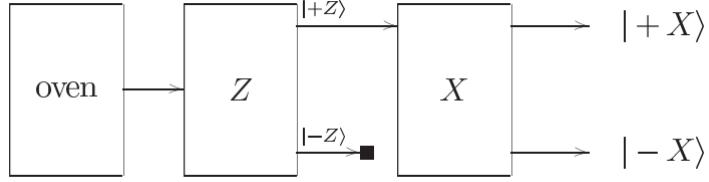


Figure 1.4: Cascaded Stern-gerlach Experiment

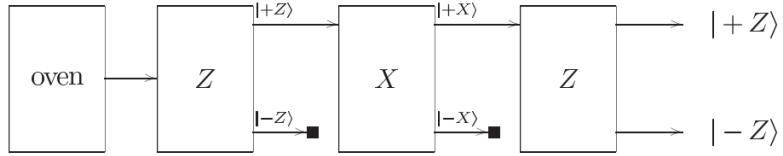


Figure 1.5: Three stage cascaded Stern-Gerlach measurements

and $| -X \rangle$ states, and a $| +X \rangle$ state consists of equal portions of $| +Z \rangle$ and $| -Z \rangle$ states. Similar conclusions can be reached if the Stern-Gerlach apparatus is aligned along some other axis like the \hat{y} axis.

The qubit model provides a simple explanation of this experimentally observed behavior. Let $|0\rangle$ and $|1\rangle$ be the states of a qubit, and make the assignments.

$$\begin{aligned} |+Z\rangle &\leftarrow |0\rangle \\ |-Z\rangle &\leftarrow |1\rangle \\ |+X\rangle &\leftarrow (|0\rangle + |1\rangle)/\sqrt{2} \\ |-X\rangle &\leftarrow (|0\rangle - |1\rangle)/\sqrt{2} \end{aligned}$$

then the results of the cascaded Stern-Gerlach experiment can be explained by assuming that the \hat{z} Stern-Gerlach apparatus measures the spin (that is, the qubit) in the computational $|0\rangle, |1\rangle$, and the \hat{x} Stern-Gerlach apparatus measures the spin with respect to the basis $(|0\rangle + |1\rangle)/\sqrt{2}, (|0\rangle - |1\rangle)/\sqrt{2}$. For example, in the cascaded $\hat{z} - \hat{x} - \hat{z}$ experiment, if we assume that the spin are in the state $|+Z\rangle = |0\rangle = (|+X\rangle + |-X\rangle)/\sqrt{2}$ after exiting the first Stern-gerlach experiment, then the probability for obtaining $|+X\rangle$ out of the second apparatus is $1/2$, and the probability for $| -X \rangle$ is $1/2$. Similarly, the probability for obtaining $|+Z\rangle$ out of the third apparatus is $1/2$. A qubit model thus properly predicts results from this type of cascaded Stern-Gerlach experiment.

This example demonstrates how qubits could be a believable way of modelling systems in Nature. Of course it doesn't establish beyond all doubt that the qubit model is the correct way of understanding electron spin - far more experimental corroboration is required. nevertheless, because of many experiments like these, we now believe that the qubit model (and generalizations of it to higher dimensions; quantum mechanics in other words) is capable of describing every physical system.

1.7 Qiskit Examples

In this section we present some examples of Qiskit implementations of the concepts introduced in this lesson.

1.7.1 Vector and Matrices in Python

Qiskit uses the Python programming language, so before discussing Qiskit specifically it may be helpful to very briefly discuss matrix and vector computations in Python. In Python, matrix and vector computations can be performed using the array class from the NumPy library (which includes many additional components for numerical computation).

Here is an example of a code cell that defines two vectors, ket0 and ket1, corresponding to the qubit state vectors $|0\rangle$ and $|1\rangle$ and displays their average.

```
1 from numpy import array
2
3 ket0 = array([1, 0])
4 ket1 = array([0, 1])
5
6 display(ket0 / 2 + ket1 / 2)
```

Output:

```
1 array ([0.5, 0.5])
```

It is not actually necessary to explicitly use the display command to see the result of this computation. We may instead simply write the expression of interest as the last line of the code cell, and it will be returned as its output:

```
1 ket0 / 2 + ket1 / 2
```

Output:

```
1 array ([0.5, 0.5])
```

This code cell also illustrates that running code cells sequentially on a given page of this textbook has a cumulative effect, so it is not necessary to reload the array class or define ket0 and ket1 again. Reloading the page or switching to another page will, however, reset everything to its initial state.

As a general guideline, code cells within each subsection of this course are intended to be run sequentially. So, if running a code cell generates an error, be sure to first run all previous code cells within the subsection in which that code cell appears.

We can also use array to create matrices that represent operations.

```

1 M1 = array ([[1 ,  1], [0 ,  0]])
2 M2 = array ([[1 ,  1], [1 ,  0]])
3
4 M1 / 2 + M2 / 2

```

Output:

```

1 array ([[1. ,  1. ],
2         [0.5,  0. ]])

```

Matrix multiplication (including matrix-vector multiplication as a special case) can be performed using the matmul function from NumPy :

```

1 from numpy import matmul
2
3 display(matmul(M1, ket1))
4 display(matmul(M1, M2))
5 display(matmul(M2, M1))

```

Output:

```

1 array ([1 ,  0])
2 array ([[2 ,  1],
3         [0 ,  0]])
4 array ([[1 ,  1],
5         [1 ,  1]])

```

1.7.2 States, measurements and operations

Qiskit includes several classes that allow for states, measurements, and operations to be easily created and manipulated — so starting from scratch and programming everything that is needed to simulate quantum states, measurements, and operations in Python is not required. Some examples to get started are included below.

Defining and displaying state vectors

Qiskit's Statevector class provides functionality for defining and manipulating quantum state vectors. The following code cell imports the Statevector class and defines a few vectors using it. (Note that we need the sqrt function from the NumPy library to compute the square roots for the vector u.)

```

1 from qiskit.quantum_info import Statevector
2 from numpy import sqrt
3
4 u = Statevector([1 / sqrt(2), 1 / sqrt(2)])
5 v = Statevector([(1 + 2.0j) / 3, -2 / 3])
6 w = Statevector([1 / 3, 2 / 3])
7
8 print("State vectors u, v, and w have been defined.")

```

Output:

```
1 State vectors u, v, and w have been defined.
```

The Statevector class provides a draw method for displaying state vectors, including latex and text options for different visualizations, as this code cell demonstrates:

```

1 display(u.draw("latex"))
2 display(v.draw("text"))

```

Output:

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

$$[0.33333333 + 0.66666667j, -0.66666667 + 0.j]$$

The Statevector class also includes the is_valid method, which checks to see if a given vector is a valid quantum state vector (i.e., that it has Euclidean norm equal to 1):

```

1 display(u.is_valid())
2 display(w.is_valid())

```

Output:

```

1 True
2
3 False

```

Simulating Measurements using Statevector

Next we will see one way that measurements of quantum states can be simulated in Qiskit, using the measure method from the Statevector class.

First, we create a qubit state vector v and then display it.

```

1 v = Statevector([(1 + 2.0j) / 3, -2 / 3])
2 v.draw("latex")

```

Output:

$$\left(\frac{1}{3} + \frac{2\iota}{3}\right) |0\rangle - \frac{2}{3} |1\rangle$$

Next, running the measure method simulates a standard basis measurement. It returns the result of that measurement, plus the new quantum state of our system after that measurement

```

1 v.measure()

```

Output:

```

1 ('1',
2 Statevector([ 0.+0.j, -1.+0.j],
3 dims=(2,)))

```

Measurement outcomes are probabilistic, so the same method can return different results. Try running the cell a few times to see this.

For the particular example of the vector `v`, defined above, the measure method defines the quantum state vector after the measurement takes place to be

$$\frac{1 + 2\iota}{\sqrt{5}} |0\rangle$$

(rather than $|0\rangle$) or

$$-|1\rangle$$

(rather than $|1\rangle$), depending on the measurement outcome. In both cases, the alternatives are, in fact, equivalent - they are said to differ by a global phase because one is equal to the other multiplied by a complex number on the unit circle. This issue is discussed in greater detail in the next chapter.

As an aside, `Statevector` will throw an error if the measure method is applied to an invalid quantum state vector. Feel free to give it a try if you are interested in seeing what an error looks like.

`Statevector` also comes with a `sample_counts` method that allows for the simulation of any number of measurements on the system. For example, the following cell shows the outcome of measuring the vector `v` 1000 times, which (with high probability) results in the outcome 0 approximately 5 out of every 9 times (or about 556 of the 1000 trials) and the outcome 1 approximately 4 out of every 9 times (or about 444 out of the 100 trials). The cell also demonstrates the `plot_histogram` function for visualizing the results.

```

1 from qiskit.visualization import plot_histogram
2
3 statistics = v.sample_counts(1000)
4 display(statistics)
5 plot_histogram(statistics)

```

Output:

```

1 {np.str_('0'): np.int64(568), np.str_('1'): np.int64(432)}

```

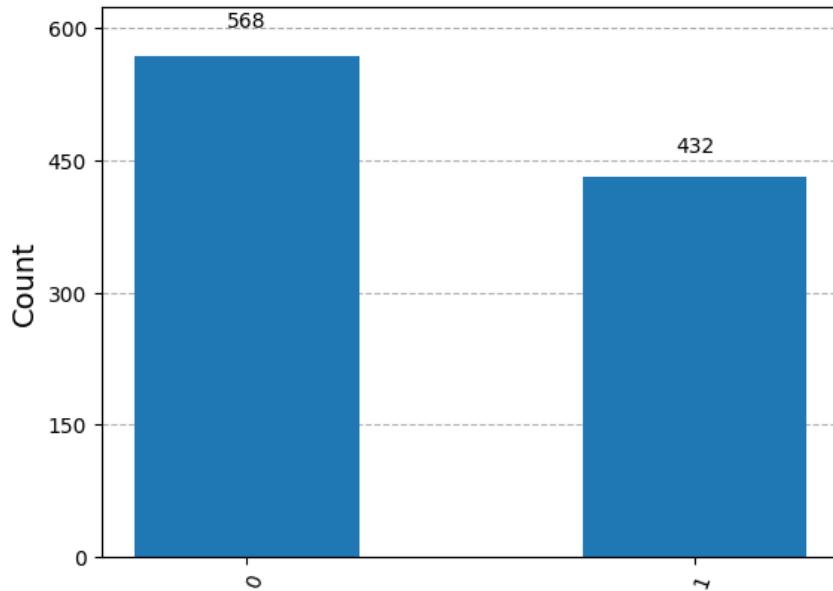


Figure 1.6: Histogram of measurement outcomes

Running the cell multiple times and trying different numbers of samples in place of 1000 may be helpful for developing some intuition for how the number of trials influences the estimated probabilities.

Performing operations with Operator and Statevector

Unitary operations can be defined and performed on state vectors in Qiskit using the operator class, as in the example that follows:

```

1 from qiskit.quantum_info import Operator
2
3 X = Operator([[0, 1], [1, 0]])
4 Y = Operator([[0, -1.0j], [1.0j, 0]])

```

```

5 Z = Operator ([[1 , 0] , [0 , -1]])
6 H = Operator ([[1 / sqrt(2) , 1 / sqrt(2)] , [1 / sqrt(2) , -1 / sqrt(2)]])
7 S = Operator ([[1 , 0] , [0 , 1.0j]])
8 T = Operator ([[1 , 0] , [0 , (1 + 1.0j) / sqrt(2)]])
9
10 v = Statevector ([1 , 0])
11
12 v = v.evolve (H)
13 v = v.evolve (T)
14 v = v.evolve (H)
15 v = v.evolve (T)
16 v = v.evolve (Z)
17
18 v.draw ("text")

```

Output:

```
1 [ 0.85355339+0.35355339j , -0.35355339+0.14644661j ]
```

Looking ahead toward Quantum Circuits

Quantum circuits won't be formally introduced until the next few chapters, but we can nevertheless experiment with composing qubit unitary operations using Qiskit's QuantumCircuit class. In particular, we may define a quantum circuit (which in this case will simply be a sequence of unitary operations performed on a single qubit)

```

1 from qiskit import QuantumCircuit
2
3 circuit = QuantumCircuit (1)
4
5 circuit.h(0)
6 circuit.t(0)
7 circuit.h(0)
8 circuit.t(0)
9 circuit.z(0)
10
11 circuit.draw()

```

Output: The operations are applied sequentially, starting on the left and ending on the right in the figure. Let us first initialize a starting quantum state vector and then evolve that state according to the sequence of operations.

```

1 ket0 = Statevector ([1 , 0])
2 v = ket0.evolve (circuit)
3 v.draw ("text")

```

Output:



Figure 1.7: Quantum Circuit

```

1 [ 0.85355339+0.35355339j, -0.35355339+0.14644661j ]

```

Finally, let's simulate the result of running this experiment (i.e., preparing the state $|0\rangle$, applying the sequence of operations represented by the circuit, and measuring 4000 times.

```

1 statistics = v.sample_counts(4000)
2 plot_histogram(statistics)

```

Output: In the previous lesson, we learned about Qiskit's Statevector and Operator

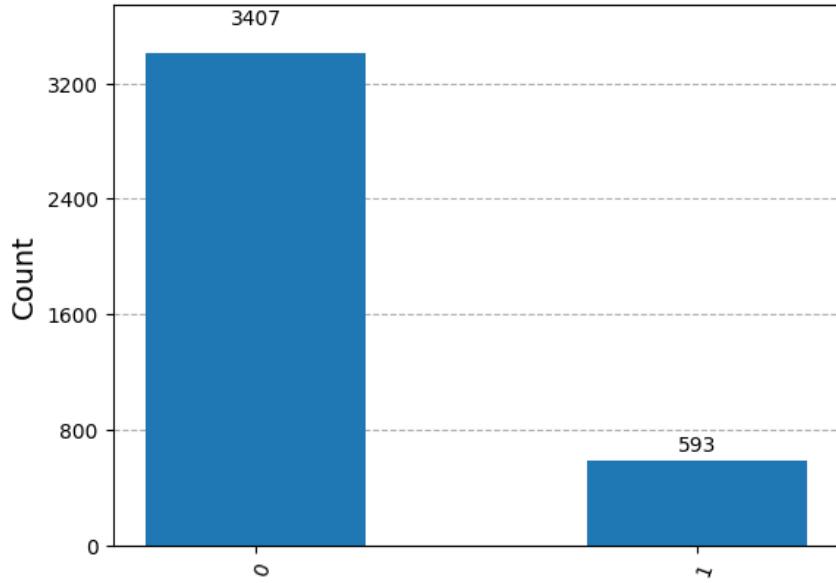


Figure 1.8: Histogram of measurement outcomes

classes, and used them to simulate quantum systems. In this section, we'll use them to explore the behavior of multiple systems. We'll start by importing these classes, as well as the square root function from NumPy .

```

1 from qiskit.quantum_info import Statevector, Operator
2 from numpy import sqrt

```

Tensor products

The Statevector class has a tensor method which returns the tensor product of itself and another Statevector .

For example, below we create two state vectors representing $|0\rangle$ and $|1\rangle$, and use the tensor method to create a new vector, $|0\rangle \otimes |1\rangle$.

```

1 zero, one = Statevector.from_label("0"), Statevector.from_label("1")
2 zero.tensor(one).draw("latex")

```

Output:

$$|01\rangle$$

In another example below, we create state vectors representing the $|+\rangle$ and $\frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$ states, and combine them to create a new state vector. We'll assign this new vector to the variable psi.

```

1 plus = Statevector.from_label("+")
2 i_state = Statevector([1 / sqrt(2), 1j / sqrt(2)])
3 psi = plus.tensor(i_state)
4
5 psi.draw("latex")

```

Output:

$$\frac{1}{2}|00\rangle + \frac{i}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{i}{2}|11\rangle$$

The Operator class also has a tensor method. In the example below, we create the X and I gates and display their tensor product.

```

1 X = Operator([[0, 1], [1, 0]])
2 I = Operator([[1, 0], [0, 1]])
3
4 X.tensor(I)

```

Output:

```

1 Operator([[0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j],
2           [0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j],
3           [1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j],
4           [0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j]],
5           input_dims=(2, 2), output_dims=(2, 2))

```

We can then treat these compound states and operations as we did single systems in the previous lesson. For example, in the cell below we calculate $(I \otimes X)|\psi\rangle$ for the state psi we defined above. (The operator shown tensors matrices together.)

```
1 psi.evolve(I ^ X).draw("latex")
```

$$\frac{\iota}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{\iota}{2}|10\rangle + \frac{1}{2}|11\rangle$$

Below, we create a CX operator and calculate $CX|\psi\rangle$.

```
1 CX = Operator(
2     [
3         [1, 0, 0, 0],
4         [0, 1, 0, 0],
5         [0, 0, 0, 1],
6         [0, 0, 1, 0],
7     ]
8 )
9
10 psi.evolve(CX).draw("latex")
```

$$\frac{1}{2}|00\rangle + \frac{\iota}{2}|01\rangle + \frac{\iota}{2}|10\rangle + \frac{1}{2}|11\rangle$$

1.7.3 Partial measurements

In the previous page, we used the measure method to simulate a measurement of the quantum state vector. This method returns two items: the simulated measurement result, and the new Statevector given this measurement.

By default, measure measures all qubits in the state vector, but we can provide a list of integers to only measure the qubits at those indices. To demonstrate, the cell below creates the state $W = (|001\rangle + |001\rangle + |010\rangle + |100\rangle)$ (Note that Qiskit is primarily designed for use with qubit-based quantum computers. As such, Statevector will try to interpret any vector with elements as a system of n qubits. You can override this by passing a dims argument to the constructor. For example, dims=(4,2) would tell Qiskit the system has one four-level system, and one two-level system (qubit).)

```
1 W = Statevector([0, 1, 1, 0, 1, 0, 0, 0] / sqrt(3))
2 W.draw("latex")
```

$$\frac{\sqrt{3}}{3}|001\rangle + \frac{\sqrt{3}}{3}|010\rangle + \frac{\sqrt{3}}{3}|100\rangle$$

The cell below simulates a measurement on the rightmost qubit (which has index 0). The other two qubits are not measured.

```
1 result , new_sv = W.measure([0]) # measure qubit 0
2 print(f"Measured: {result}\nState after measurement:")
3 new_sv.draw("latex")
```

Output:

```
1 Measured: 1
2 State after measurement:
```

$$|001\rangle$$

Try running the cell a few times to see different results. Notice that measuring a 1 means that we know both the other qubits are $|0\rangle$, but measuring a 0 means the remaining two qubits are in the state $\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$.

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 α_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 α_{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 α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$ (since its a unit vector). The condition $|\alpha|^2 + |\beta|^2 = 1$ is called the *normalization condition*. The complex numbers α and β 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 α and β 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 α_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 Hermitian and as proved in the appendix we know that e^A is unitary for any Hermitian 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 description of the dynamics using unitary operators. Since the Hamiltonian H is Hermitian, it is normal and hence has spectral decomposition:

$$H = \sum_E E |E\rangle \langle E|$$

where all E are real by the Hermicity of H , and $|E\rangle$ is an orthonormal basis of the Hilbert space. We then have that:

$$\begin{aligned} U(t_1, t_2) &= \exp\left(\frac{-iH(t_2-t_1)}{\hbar}\right) = \exp\left(\frac{-i\sum_E E |E\rangle \langle E| (t_2-t_1)}{\hbar}\right) \\ &= \sum_E \exp\left(\frac{-iE(t_2-t_1)}{\hbar}\right) |E\rangle \langle E| \end{aligned}$$

Hence calculating $U^\dagger(t_1, t_2)$ we have

$$\begin{aligned} U^\dagger(t_1, t_2) &= \left(\sum_E \exp\left(\frac{-iE(t_2-t_1)}{\hbar}\right) |E\rangle \langle E| \right)^\dagger \\ &= \sum_E \left(\exp\left(\frac{iE(t_2-t_1)}{\hbar}\right) \right)^* (|E\rangle \langle E|)^\dagger \\ &= \sum_E \exp\left(\frac{iE(t_2-t_1)}{\hbar}\right) |E\rangle \langle E| \end{aligned}$$

Therefore computing $U^\dagger(t_1, t_2)U(t_1, t_2)$ we have:

$$\begin{aligned}
U^\dagger(t_2, t_1)U(t_2, t_1) &= \left(\sum_E \exp\left(\frac{-\imath E(t_2 - t_1)}{\hbar}\right) |E\rangle\langle E| \right) \left(\sum_{E'} \exp\left(\frac{\imath E'(t_2 - t_1)}{\hbar}\right) |E'\rangle\langle E'| \right) \\
&= \sum_E \sum_{E'} \exp\left(\frac{-\imath E(t_2 - t_1)}{\hbar}\right) \exp\left(\frac{\imath E(t_2 - t_1)}{\hbar}\right) |E\rangle\langle E| \\
&= \sum_E \exp\left(\frac{-\imath E(t_2 - t_1)}{\hbar}\right) \exp\left(\frac{\imath E(t_2 - t_1)}{\hbar}\right) |E\rangle\langle E| \\
&= \sum_E |E\rangle\langle E| \\
&= I
\end{aligned}$$

where in the second equality we use the fact that the eigenstates are orthogonal. We conclude that U is unitary. For the purpose of Quantum Computing, we will deal with discrete time evolution of the quantum system described by the exponential 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 $|\psi(t_2)\rangle = U(t_2, t_1)|\psi(t_1)\rangle$.

Example 2.2.1. Use the spectral decomposition to show that $K = -\imath \log(U)$ is Hermitian for any unitary U , and thus $U = \exp(\imath K)$ for some Hermitian K .

Suppose U is unitary. Then, U is normal and hence has spectral decomposition:

$$U = \sum_j \lambda_j |j\rangle\langle j|$$

where $|j\rangle$ are the eigenvectors of U with eigenvalues λ_j , and $|j\rangle$ forms an orthonormal basis of the Hilbert space. Recall that all the eigenvalues of unitary operators have eigenvalues of modulus 1, so we can let $\lambda_j = \exp(\imath\theta_j)$ where $\theta_j \in \mathbb{R}$ and hence write the above as:

$$U = \sum_j \exp(\imath\theta_j) |j\rangle\langle j|$$

We then have that:

$$K = -\imath \log(U) = -\log \left(\sum_j \exp(\imath\theta_j) |j\rangle\langle j| \right) = \sum_j -\imath \log(\exp(\imath\theta_j)) |j\rangle\langle j| = \sum_j -\imath(\imath\theta_j) |j\rangle\langle j|$$

$$= \sum_j \theta_j |j\rangle \langle j|$$

We then observe that:

$$K^\dagger = \left(\sum_j \theta_j |j\rangle \langle j| \right)^\dagger = \sum_j \theta_j |j\rangle \langle j|$$

as the θ_j s are real and $(|j\rangle \langle j|)^\dagger = |j\rangle \langle j|$. Hence K is Hermitian. Then, multiplying both sides in $K = -\iota \log(U)$ by ι and exponentiating both sides, we obtain the desired relation.

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 Schrodinger's equation with a time varying Hamiltonian, to some good approximation. Thus we will describe the evolution 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 α_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, α_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 α_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:

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

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

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

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

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

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

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

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.

Case I: 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 $|\psi_i\rangle$.

Case II: 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 (non-zero) component parallel to $|\psi_1\rangle$, and a component orthogonal to $|\psi_1\rangle$.

Example 2.3.3. Suppose such a measurement is possible that two non-orthogonal states $|\psi_1\rangle$ and $|\psi_2\rangle$ can be distinguished. Defining $E_i \equiv \sum_{j:f(j)=i} M_j M_j^\dagger$, these observations may be written as:

$$\langle\psi_1|E_1|\psi_1\rangle = 1; \quad \langle\psi_2|E_2|\psi_2\rangle = 1$$

Since $\sum_i E_i = I$ it follows that $\sum_i \langle\psi_1|E_i|\psi_1\rangle = 1$, and since $\langle\psi_1|E_1|\psi_1\rangle = 1$ we must have $\langle\psi_1|E_2|\psi_1\rangle = 0$, and thus $\sqrt{E_2}|\psi_1\rangle = 0$. Suppose we decompose $|\psi_2\rangle = \alpha|\psi_1\rangle + \beta|\psi\rangle$, where $|\psi\rangle$ is orthonormal to $|\psi_1\rangle$, $|\alpha|^2 + |\beta|^2 = 1$, and $|\beta| < 1$ since $|\psi_1\rangle$ and $|\psi_2\rangle$ are not orthogonal. Then $\sqrt{E_2}|\psi_2\rangle = \beta\sqrt{E_2}|\psi\rangle$, which implies a contradiction as

$$\langle\psi_2|E_2|\psi_2\rangle = |\beta|^2 \langle\psi|E_2|\psi\rangle \leq |\beta|^2 < 1$$

where the second last inequality follows from the observation that

$$\langle\psi|E_2|\psi\rangle \leq \sum_i \langle\psi|E_i|\psi\rangle = \langle\psi|\psi\rangle = 1$$

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 δ_{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 choose 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 2.3.4. 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 2.3.5. 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 of the basis vectors $|\phi_j\rangle$ then the measurement gives outcome j with probability 1.

Example 2.3.6. 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, its 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 measure-

ment is:

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

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-measure (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 measurement on a slightly larger space that yields the exact probability distribution over measurement outcomes (Neumark's theorem)*

Example 2.3.7. 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 measurement with three possible outcomes: 0, 1 and 2, such that:

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

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

This cannot be achieved with projective measurement on the qubit, but can be achieved with following 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 state $|0\rangle$ 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.

Example 2.3.8. (Cascaded measurements are single measurements) Suppose $\{L_l\}$ and $\{M_m\}$ are two sets of measurement operators. Show that a measurement defined by the measurement operators $\{L_l\}$ followed by a measurement defined by the measurement operators $\{M_m\}$ is physically equivalent to a single measurement defined by measurement operators $\{N_{lm}\}$ with the representation $N_{lm} = M_m L_l$

Suppose we have (normalized) initial quantum state $|\psi_0\rangle$. Then, the state after measurement of L_l is given by definition to be:

$$|\psi_0\rangle \rightarrow |\psi_1\rangle = \frac{L_l |\psi_0\rangle}{\sqrt{\langle\psi_0| L_l^\dagger L_l |\psi_0\rangle}}$$

The state after measurement of M_m on $|\psi_1\rangle$ is then given to be:

$$|\psi_1\rangle \rightarrow |\psi_2\rangle = \frac{M_m |\psi_1\rangle}{\sqrt{\langle\psi_1| M_m^\dagger M_m |\psi_1\rangle}}$$

$$\begin{aligned}
& M_m \left(\frac{L_l |\psi_0\rangle}{\sqrt{\langle\psi_0|L_l^\dagger L_l|\psi_0\rangle}} \right) \\
&= \frac{\sqrt{\left(\frac{L_l^\dagger \langle\psi_0|}{\sqrt{\langle\psi_0|L_l^\dagger L_l|\psi_0\rangle}} \right) M_m^\dagger M_m \left(\frac{L_l |\psi_0\rangle}{\sqrt{\langle\psi_0|L_l^\dagger L_l|\psi_0\rangle}} \right)}} \\
&= \frac{M_m L_l |\psi_0\rangle}{\sqrt{\langle\psi_0|L_l^\dagger L_l|\psi_0\rangle}} \frac{\sqrt{\langle\psi_0|L_l^\dagger L_l|\psi_0\rangle}}{\sqrt{\langle\psi_0|L_l^\dagger M_m^\dagger M_m L_l^\dagger|\psi_0\rangle}} \\
&= \frac{M_m L_l |\psi_0\rangle}{\sqrt{\langle\psi_0|L_l^\dagger M_m^\dagger M_m L_l^\dagger|\psi_0\rangle}}
\end{aligned}$$

We see that $|\psi_2\rangle = |\psi_3\rangle$ (that is, the cascaded measurement produces the same result as the single measurement), proving the claim.

Example 2.3.9. Show that any measurement where the measurement operator and the POVM elements coincide is a projective measurement.

Suppose we have that the measurement operators are equal to the POVM elements E_m . In this case we have that:

$$M_m = E_m = M_m^\dagger M_m$$

Recall that $M_m^\dagger M_m$ is positive, so it follows that M_m is positive and hence Hermitian. Hence $M_m^\dagger = M_m$ and therefore:

$$M_m = M_m^\dagger M_m = M_m^2$$

From which we conclude that M_m are projective measurement operators.

Example 2.3.10. Suppose a measurement is described by measurement operators M_m . Show that there exist unitary operators U_m such that $M_m = U_m \sqrt{E_m}$, where E_m is the POVM associated to the measurement.

Since M_m is a linear operator, by the left polar decomposition there exists unitary U such that:

$$M_m = U \sqrt{M_m^\dagger M_m} = U \sqrt{E_m}$$

where in the last equality we use that $M_m^\dagger M_m = E_m$.

Example 2.3.11. Suppose Bob is given a quantum state chosen from a set $|\psi_1\rangle, |\psi_2\rangle, \dots, |\psi_m\rangle$ of linearly independent states. Construct a POVM $\{E_1, E_2, \dots, E_{m+1}\}$ such that if outcome E_i occurs, $1 \leq i \leq m$, then Bob knows with certainty that he was given the state $|\psi_i\rangle$. (The POVM must be such that $\langle\psi_i|E_i|\psi_i\rangle > 0$ for each i).

Let \mathcal{H} be the Hilbert space where the given states lie, and let V be the m -dimensional subspace spanned by $|\psi_1\rangle, \dots, |\psi_m\rangle$. For each $i \in \{1, \dots, m\}$, let W_i be the subspace of V spanned by $\{|\psi_j\rangle : j \neq i\}$. Let W_i^\perp be the orthogonal complement of W_i which consists of all states in \mathcal{H} orthogonal to all states in W_i . We then have that any vector in V can be written as the sum of a vector in W_i and $W_i^\perp \cap V$. Therefore, for any $|\psi_i\rangle$ we can write:

$$|\psi_i\rangle = |w_i\rangle + |p_i\rangle$$

where $|w_i\rangle \in W_i$ and $|p_i\rangle \in W_i^\perp \cap V$. Define $E_i = \frac{|p_i\rangle\langle p_i|}{m}$. By construction, we have that for any $|\psi\rangle \in \mathcal{H}$:

$$\langle\psi|E_i|\psi\rangle = \frac{|\langle\psi|p_i\rangle|^2}{m} \geq 0$$

so that E_i s are positive as required. Furthermore, defining $E_{i+1} = I - \sum_{i=1}^m E_i$, we again see that for any $|\psi\rangle \in \mathcal{H}$:

$$\langle\psi|E_{i+1}|\psi\rangle = \langle\psi|I|\psi\rangle - \sum_{i=1}^m \langle\psi|E_i|\psi\rangle = 1 - \sum_{i=1}^m \langle\psi|E_i|\psi\rangle \geq 1 - \sum_{i=1}^m \frac{1}{m} = 0$$

so E_{i+1} is also positive as required. Finally, to see that E_1, \dots, E_m have the desired properties, observe by construction that since $|p_i\rangle \in W_i^\perp \cap V$, it follows that $\langle\psi_j|p_i\rangle = 0$ for any $j \neq i$ (as the $|p_i\rangle$), we observe that:

$$\langle\psi_i|E_i|\psi_i\rangle = (\langle w_i| + \langle p_i|) \frac{|p_i\rangle\langle p_i|}{m} (|w_i\rangle + |p_i\rangle) = \frac{|\langle p_i|p_i\rangle|^2}{m} = \frac{1}{m} \geq 0$$

so if Bob measures E_i , he can be certain the he was given the state $|\psi_i\rangle$.

Example 2.3.12. Express the states $(|0\rangle + |1\rangle)/\sqrt{2}$ and $(|0\rangle - |1\rangle)/\sqrt{2}$ in a basis in which they are not the same up to a relative phase shift.

Let us define our basis to be $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$. Our two states are then just the basis vectors of this basis $(|+\rangle, |-\rangle)$ and are not the same up to relative phase shift.

Example 2.3.13. Suppose we prepare a quantum system in an eigenstate $|\psi\rangle$ of some observable M with corresponding eigenvalue m . What is the average observed value of M , and the standard deviation?

By definition of expectation, we have that:

$$\langle \psi | M | \psi \rangle = \langle \psi | m | \psi \rangle = m \langle \psi | \psi \rangle = m$$

Similarly, the standard deviation is given as:

$$\begin{aligned}\Delta(M) &= \sqrt{\langle \psi | M^2 | \psi \rangle - (\langle \psi | M | \psi \rangle)^2} \\ &= \sqrt{\langle \psi | Mm | \psi \rangle - (\langle \psi | m | \psi \rangle)^2} \\ &= \sqrt{\langle \psi | m^2 | \psi \rangle - (m \langle \psi | \psi \rangle)^2} \\ &= \sqrt{m^2 - (m)^2} \\ &= 0\end{aligned}$$

Example 2.3.14. Suppose we have qubit in the state $|0\rangle$, and we measure the observable X . What is the average value of X ? What is the standard deviation of X ?

By the definition of expectation we have:

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

Similarly, for standard deviation, we have:

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

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:

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

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

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

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

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 probability 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 probability 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 probability 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 $|\psi\rangle = \frac{1}{2}|00\rangle - \frac{\iota}{2}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$. 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 $|-\frac{\iota}{2}|^2 + |\frac{1}{\sqrt{2}}|^2 = \frac{3}{4}$. 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:

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

Example 2.4.5. Show that the average value of the observable X_1Z_2 for a two qubit system measured in the state $(|00\rangle + |11\rangle)/\sqrt{2}$ is zero.

$$\begin{aligned} \langle X_1Z_2 \rangle &= \left(\frac{\langle 00| + \langle 11|}{\sqrt{2}} \right) X_1Z_2 \left(\frac{|00\rangle + |11\rangle}{\sqrt{2}} \right) \\ &= \left(\frac{\langle 00| + \langle 11|}{\sqrt{2}} \right) \left(\frac{X_1Z_2|00\rangle + X_1Z_2|11\rangle}{\sqrt{2}} \right) \\ &= \left(\frac{\langle 00| + \langle 11|}{\sqrt{2}} \right) \left(\frac{|10\rangle - |01\rangle}{\sqrt{2}} \right) \\ &= \frac{1}{2}(\langle 00|10\rangle - \langle 00|01\rangle + \langle 11|10\rangle - \langle 11|01\rangle) \\ &= \frac{1}{2}(0 + 0 + 0 + 0) \\ &= 0 \end{aligned}$$

Example 2.4.6. Suppose V is a Hilbert space with a subspace W . Suppose $U : W \rightarrow V$ is a linear operator which preserves inner products, that is, for any $|w_1\rangle$ and $|w_2\rangle$ in W ,

$$\langle w_1|U^\dagger U|w_2\rangle = \langle w_1|w_2\rangle$$

Prove that there exists a unitary operator $U' : V \rightarrow V$ which extends U . That is, $U'|w\rangle = U|w\rangle$ for all $|w\rangle \in W$, but U' is defined on the entire space V . Usually we omit the prime symbol ' and just write U to denote the extension.

By assumption we have that U is unitary on W as $\langle w_1|U^\dagger U|w_2\rangle = \langle w_1|w_2\rangle$ and hence $U^\dagger U = I_W$. Hence, it has spectral decomposition:

$$U = \sum_j \lambda_j |j\rangle \langle j|$$

where $\{|j\rangle\}$ is an orthonormal basis of the subspace W . Then, let $\{|j\rangle\} \cup \{|i\rangle\}$ be an orthonormal basis of the full space V . We then define:

$$U^\dagger = \sum_j \lambda_j |j\rangle \langle j| + \sum_i |i\rangle \langle i| = U + \sum_i |i\rangle \langle i|$$

We can then see that for any $|w\rangle \in W$ that:

$$U' |w\rangle = \left(U + \sum_i |i\rangle \langle i| \right) |w\rangle = U |w\rangle + \sum_j |i\rangle \langle i| w \rangle = U |w\rangle$$

where in the last line we use that $\langle i|w\rangle = 0$ as $|i\rangle$ are not in the subspace W . Finally, verifying the unitarity of U' we have that:

$$\begin{aligned} U'^\dagger U' &= \left(\sum_j \lambda_j^* |j\rangle \langle j| + \sum_i |i\rangle \langle i| \right) \left(\sum_{j'} \lambda_j |j\rangle \langle j| + \sum_{i'} |i\rangle \langle i| \right) \\ &= \sum_j \sum_{j'} |j\rangle \langle j| |j'\rangle \langle j'| + \sum_j \sum_{i'} |j\rangle \langle j| |i'\rangle \langle i'| + \sum_i \sum_{j'} |i\rangle \langle i| |j'\rangle \langle j'| + \sum_i \sum_{i'} |i\rangle \langle i| |i'\rangle \langle i'| \\ &= \sum_j \sum_{j'} \langle j|j'\rangle \delta_{jj'} + \sum_i \sum_{i'} \langle i|i'\rangle \delta_{ii'} \\ &= \sum_j |j\rangle \langle j| + \sum_i |i\rangle \langle i| \\ &= I \end{aligned}$$

Chapter 3

Qubits and quantum gates

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. In this chapter we explore quantum computation in detail. This chapter develops the fundamental principles of quantum computation, and establishes the basic building blocks for quantum circuits a universal language for describing sophisticated quantum computations. This chapter will introduce the fundamental model of quantum computation, the quantum circuit model and the concept that there exists a small set of gates which are universal, that is, any quantum computation whatsoever can be expressed in those gates along with some basic results of quantum computation.

Many interesting problems are impossible to solve on a classical computer, not because they are in principle insoluble, but because of the astronomical resources required to solve realistic cases of the problem. The spectacular promise of quantum computers is to enable new algorithms which render feasible problems requiring exorbitant resources for their solution on a classical computer. Two broad classes of quantum algorithms are known which fulfill this promise. The first class of algorithms is based upon Shor's quantum Fourier transform, and includes remarkable algorithms for solving the factoring and discrete logarithm problems, providing an exponential speedup over the best known classical algorithms. The second class of algorithms is based upon Grover's algorithm for performing quantum searching. These probes a less stringent but

Why are there so few quantum algorithms known which are better than their classical counterparts? The answer is that coming up with good quantum algorithm seems to be a difficult problem. There are at least two reasons for this. First, algorithm design, be it classical or quantum is not easy. The history of algorithms shows us that considerable ingenuity is often required to come up with near optimal algo-

rithms, even for apparently simple problems, like the multiplication of two numbers. Finding good quantum algorithm is made doubly difficult because of the additional constraint that we want our quantum algorithms to be better than the best known classical algorithm. A second reason for the difficult for finding good quantum algorithms is that our intuitions are better adapted to the classical world than they are to the quantum world. If we think about our problems using our native intuition, then the algorithms we come up with are going to be classical algorithms. It takes special insights and special tricks to come up with good quantum algorithms.

To quantify the cost we use the total number of gates required, or the circuit depth. The circuit language also comes with a toolbox of tricks that simplifies algorithm design and provides ready conceptual understanding.

3.1 Phase

‘Phase’ is a commonly used term in quantum mechanics, with several different meanings dependent upon context. At this point it is convenient to review a couple of these meaning. Consider, for example, the state $e^{i\theta} |\psi\rangle$, where $|\psi\rangle$ is a state vector, and θ is a real number. We say that the state $e^{i\theta}$ is equal to $|\psi\rangle$, up to a global phase factor $e^{i\theta}$. It is interesting to note that the statistics of measurement predicted for these two states are the same. To see this, suppose M_m is a measurement operator associated to some quantum measurement, and note that the respective probabilities for outcome m occurring are $\langle \psi | M_m^\dagger M_m | \psi \rangle$ and $\langle \psi | e^{-i\theta} M_m^\dagger M_m e^{i\theta} | \psi \rangle = \langle \psi | M_m^\dagger M_m | \psi \rangle$. Therefore, from an observational point of view these two states are identical. For this reason we may ignore global phase factors are being irrelevant to the observed properties of the physical system.

There is another kind of phase known as the relative phase, which has quite a different meaning. Consider the states

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

In the first state the amplitude of $|1\rangle$ is $\frac{1}{\sqrt{2}}$. For the second state the amplitude is $-\frac{1}{\sqrt{2}}$. In each case the magnitude of the amplitude is the same, but they differ in sign. More generally, we say that two amplitudes, a and b , differ by a relative phase if there is a real θ such that $a = \exp(i\theta)b$. More generally still, two states are said to differ by a relative phase factor in some basis if each of the amplitudes in that basis is related by such a phase factor. For example, the two states displayed above are the same up to a relative phase shift because the $|0\rangle$ amplitudes are identical

(a relative phase factor of 1), and the $|1\rangle$ amplitudes differ only by a relative phase factor of -1 . The difference between relative phase factors and global phase factors is that for the relative phase factors the phase factors may vary from amplitude to amplitude. This makes the relative phase a basis-dependent concept unlike global phase. As a result, states which differ only by relative phases in some basis give rise to physically observable differences in measurement statistics, and it is not possible to regard these states as physically equivalent, as we do with states differing by a global phase factor.

Example 3.1.1. Express the states $(|0\rangle + |1\rangle)/\sqrt{2}$ and $(|0\rangle - |1\rangle)/\sqrt{2}$ in a basis in which they are not the same up to a relative phase shift. in the $|+\rangle, |-\rangle$ basis they will not be the same even up to a relative phase shift.

3.2 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, α and β 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 α and β 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 ϕ_α and ϕ_β 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 ϕ_α 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 - θ and ϕ 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.

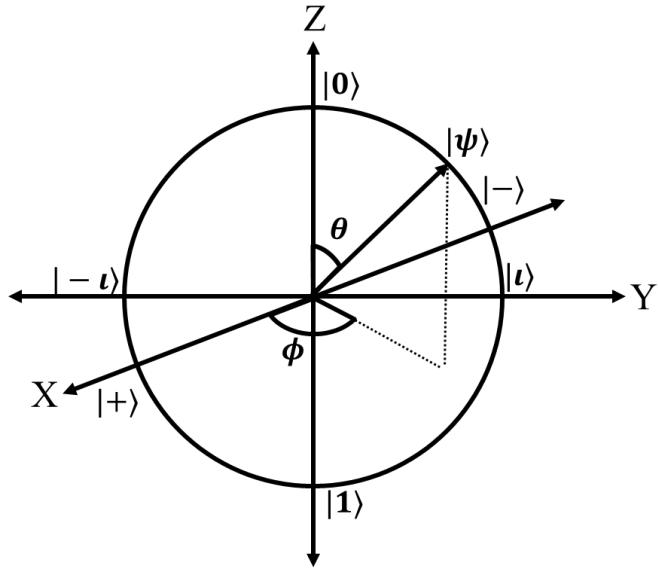


Figure 3.1: Bloch-Poincare Sphere

Thus for different values of θ and ϕ we get different points on the bloch sphere which represents the different states of a qubit. For a unique mapping from the θ and ϕ 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 text
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. 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 3.2.1. The state of a qubit for different values of θ and ϕ is shown in the table 3.1.

Quantum Gates acting on Qubits can be thought of as rotations on the Bloch Sphere with some global phase. The global phase is irrelevant

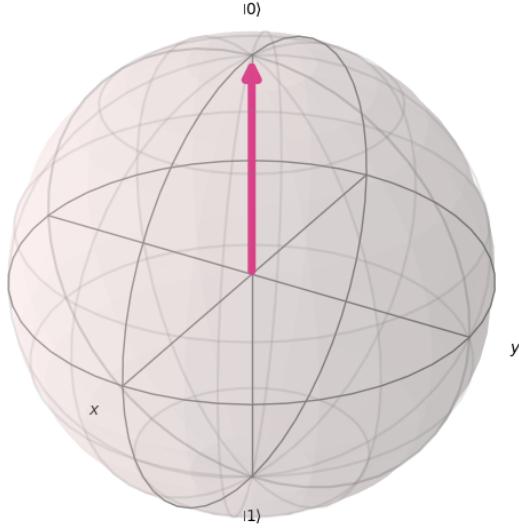


Figure 3.2: Bloch Sphere

to observable quantities and thus does not impact the practical interpretation of the rotation as a transformation on the Bloch sphere. In other words, doing matrix vector multiplication corresponding to rotation action on qubits is the same up to a global phase. A matrix-vector multiplication by a matrix corresponding to some rotation and the vector corresponding to the state of qubit will introduce a global phase shift which can be ignored. Thus, it is easier to imagine this as rotations on bloch sphere where we ignore the global phases implicitly. For example, $R_x(\pi)|0\rangle = -i|1\rangle$ using matrix vector multiplication and $R_x(\pi)|0\rangle = |1\rangle$ using the fact that rotations o bloch sphere method but both of them are the same states up to a global phase i.e. $-i = e^{i3\pi/2} \times 1$. 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:

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

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

Table 3.1: State of a Qubit for different values of θ and ϕ

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:

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

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.

3.3 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 2x2 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 3.3.1. Consider a 2×2 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.

3.3.1 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. The code for the Pauli-I Gate is

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

Table 3.2: Truth Table for Pauli-I Gate

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 In operator Notation it is:



Figure 3.3: Pauli-I Gate

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

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 π 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. In the operator Notation it is:

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

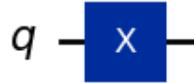


Figure 3.4: Pauli-X Gate

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

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

Its Eigen values are 1 and -1. Its 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:

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

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

Table 3.4: Truth Table for Pauli-Y Gate

Pauli-Y Gate

It is a anticlockwise π 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. 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. In the operator Notation it is:



Figure 3.5: Pauli-Y Gate

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

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:

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

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 $Y = iXZ$ or $\sigma_Y = i\sigma_X\sigma_Z$ 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 π in anticlockwise direction and is equivalent to a rotation about the X-axis by anticlockwise π followed by a rotation about the Z-axis by anticlockwise π (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 π 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. 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

```

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

Table 3.5: Truth Table for Pauli-Z Gate

```

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. In the operator Notation it is:



Figure 3.6: Pauli-Z Gate

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

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:

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

Example 3.3.2. (Pauli Matrices: Hermitian and Unitary) Show that the pauli matrices are Hermitian and unitary.

We check I, X, Y, Z in turns.

$$\begin{aligned} I^\dagger &= \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T \right)^* = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^* = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I \\ I^\dagger I &= II = I \\ X^\dagger &= \left(\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^T \right)^* = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^* = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = X \\ X^\dagger X &= XX = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = I \\ Y^\dagger &= \left(\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}^T \right)^* = \begin{bmatrix} 0 & i \\ -i & 0 \end{bmatrix}^* = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = Y \\ Y^\dagger Y &= YY = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ Z^\dagger &= \left(\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}^T \right)^* = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}^* = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = Z \\ Z^\dagger Z &= ZZ = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I \end{aligned}$$

Example 3.3.3. Calculate the matrix representation of the tensor products of the Pauli operators (a) X and Z ; (b) I and X ; (c) X and I . Is the tensor product commutative?

Using Kronecker product we have,

1.

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

2.

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

3.

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

Comparing (b) and (c), we conclude and adjoint operations distribute over the tensor product.

Example 3.3.4. (Commutation relations for the Pauli Matrices) Verify the commutation relations

$$[X, Y] = 2\iota Z; [Y, Z] = 2\iota X; [Z, X] = 2\iota Y$$

There is an elegant way of writing this using ϵ_{jkl} , the antisymmetric tensor on three indices, for which $\epsilon_{jkl} = 0$ except for $\epsilon_{123} = \epsilon_{231} = \epsilon_{312} = 1$, and $\epsilon_{321} = \epsilon_{213} = \epsilon_{132} = -1$:

$$[\sigma_j, \sigma_k] = 2\iota \sum_{l=1}^3 \epsilon_{jkl} \sigma_l$$

We verify the proposed relations via computation in the computational basis:

$$[X, Y] = XY - YX = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & -\iota \\ \iota & 0 \end{bmatrix} - \begin{bmatrix} 0 & -\iota \\ \iota & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} \iota & 0 \\ 0 & -\iota \end{bmatrix} - \begin{bmatrix} -\iota & 0 \\ 0 & \iota \end{bmatrix} = 2\iota Z$$

$$[Y, Z] = YZ - ZY = \begin{bmatrix} 0 & -\iota \\ \iota & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & -\iota \\ \iota & 0 \end{bmatrix} = \begin{bmatrix} 0 & \iota \\ \iota & 0 \end{bmatrix} - \begin{bmatrix} 0 & -\iota \\ -\iota & 0 \end{bmatrix} = 2\iota X$$

$$[Z, X] = ZX - XZ = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} - \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = 2\iota Y$$

Example 3.3.5. (Anti-commutation relations for the Pauli matrices) Verify the anti-communal relations

$$\{\sigma_i, \sigma_j\} = 0$$

where $i \neq j$ are both chosen from the set 1, 2, 3. Also verify that ($i = 0, 1, 2, 3$)

$$\sigma_i^2 = I$$

We again verify the proposed relations via computation in the computational basis:

$$\{X, Y\} = XY + YX = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & -\iota \\ \iota & 0 \end{bmatrix} + \begin{bmatrix} 0 & -\iota \\ \iota & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} \iota & 0 \\ 0 & -\iota \end{bmatrix} + \begin{bmatrix} -\iota & 0 \\ 0 & \iota \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\{Y, Z\} = YZ + ZY = \begin{bmatrix} 0 & -\iota \\ \iota & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & -\iota \\ \iota & 0 \end{bmatrix} = \begin{bmatrix} 0 & \iota \\ \iota & 0 \end{bmatrix} + \begin{bmatrix} 0 & -\iota \\ -\iota & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\{Z, X\} = ZX + XZ = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

This proves the first claim as $\{A, B\} = AB + BA = BA + AB = \{B, A\}$ and the other 3 relations are equivalent to the ones already proven. Verifying the second claim, we have:

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

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

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

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

Remark. Note that we can write this result concisely as $\{\sigma_j, \sigma_k\} = 2\delta_{jk}I$

Example 3.3.6. Show that for $j = 1, 2, 3$

$$\sigma_j \sigma_k = \delta_{jk}I + \iota \sum_{l=1}^3 \epsilon_{jkl} \sigma_l$$

Applying the previous results we get,

$$\begin{aligned} \sigma_j \sigma_k &= \frac{[\sigma_j, \sigma_k] + \{\sigma_j, \sigma_k\}}{2} \\ &= \frac{2\iota \sum_{l=1}^3 \epsilon_{jkl} \sigma_l + 2\delta_{ij}I}{2} \\ &= \delta_{jk}I + \iota \sum_{l=1}^3 \epsilon_{jkl} \sigma_l \end{aligned}$$

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 π rotation about the X-axis, Pauli-Y Gate is a π rotation about the Y-axis and Pauli-Z Gate is a π 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:

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

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 2×2 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) as can be clearly inferred from the matrices and all the Pauli matrices are Hermitian. Note here it should be presumed that the rotations are always in anticlockwise direction.

Example 3.3.7. (Pauli operators and the outer product) The Pauli matrices can be considered as operators with respect to an orthonormal basis $|0\rangle, |1\rangle$ for a

two-dimensional Hilbert space. Express each of the Pauli operators in the outer product notation

Recall that if A has matrix representation

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}$$

with respect to $|0\rangle, |1\rangle$ as the input/output bases, then we can express A in outer product notation as:

$$A = a_{00}|0\rangle\langle 0| + a_{01}|0\rangle\langle 1| + a_{10}|1\rangle\langle 0| + a_{11}|1\rangle\langle 1|$$

Furthermore, recall the representation of Pauli matrices with respect to the orthonormal basis $\text{ket}0, |1\rangle$:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

We immediately see that:

$$\begin{aligned} I &= |0\rangle\langle 0| + |1\rangle\langle 1| \\ X &= |0\rangle\langle 1| + |1\rangle\langle 0| \\ Y &= -i|0\rangle\langle 1| + i|1\rangle\langle 0| \\ Z &= |0\rangle\langle 0| - |1\rangle\langle 1| \end{aligned}$$

Example 3.3.8. (Eigendecomposition of the Pauli matrices) Find the eigenvectors, eigenvalues and diagonal representation of the Pauli matrices I,X,Y and Z. The eigenvectors and eigenvalues of I are the entire space \mathbb{C}^2 and $\lambda = 1, 1$ respectively and its diagonal representation is

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

Recall that the eigenvectors, eigenvalues of X are $|+\rangle, |-\rangle$ and $\lambda = 1, -1$ respectively. The diagonal representation will be,

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

Similarly, the eigenvectors and eigenvalue of Y are $\lambda = 1, -1$ and $|+i\rangle, |-i\rangle$ respectively, and its diagonal representation is

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

Similarly, the eigenvectors and eigenvalues of Z are $\lambda = 1, -1$ and $|0\rangle, |1\rangle$ respectively, and its diagonal representation is

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

3.3.2 Hadamard Gate

It creates a superposition of Qubits. It is a π rotation about the X+Z axis of Bloch Sphere or a $\pi/2$ rotation about the Y axis followed by a π rotation about the X axis. 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. In the operator Notation it is:



Figure 3.7: Hadamard Gate

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

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

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 eigenvalues, $\lambda_1 = 1, \lambda_2 = -1$ and the corresponding normalized eigenvectors are $\frac{1}{\sqrt{2+\sqrt{2}}} \begin{bmatrix} 1 + \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$ and $\frac{1}{\sqrt{2-\sqrt{2}}} \begin{bmatrix} -1 + \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$ respectively. 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 we can also write the Hadamard gate as

$$H = \frac{X+Z}{\sqrt{2}} = \frac{1}{\sqrt{2}} \left(\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \right) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

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:

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

for Example, explicitly writing $H^{\otimes 2}$, we get,

$$\begin{aligned} H^{\otimes 2} &= \frac{1}{\sqrt{2^2}} \sum_{x,y} (-1)^{x \cdot y} |x\rangle \langle y| \\ &= \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \end{aligned}$$

Note that here x, y are binary length 2 strings. The sum goes through all pairwise combinations of $x, y \in \{00, 01, 10, 11\}$.

Remark. Sylvester's Construction gives an interesting recursive construction of Hadamard matrices. See https://en.wikipedia.org/wiki/Hadamard_matrix. Discussion on interesting (related) open problem concerning the maximal determinant of matrices consisting of entries of 1 and -1 can be found here https://en.wikipedia.org/wiki/Hadamard%27s_maximal_determinant_problem.

Example 3.3.9. 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,

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

3.3.3 S and S^\dagger Gate

These are some specialized 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. The code for the S Gate is as follows:

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

Table 3.7: Truth Table for S Gate

```

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

```

The circuit symbol is as shown in figure 3.8. In the operator Notation it is:

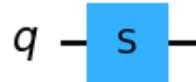


Figure 3.8: S Gate

$$S|0\rangle = |0\rangle$$

$$S|1\rangle = i|1\rangle$$

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. The code for

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

Table 3.8: Truth Table for S^\dagger Gate

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

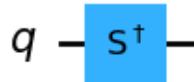
```

The circuit symbol is as shown in figure 3.9. 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:

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

Figure 3.9: S^\dagger 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:

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

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.

3.3.4 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. This is because up to an unimportant global phase T is equal to a gate which has $e^{\pm i\phi/8}$ appearing on its diagonals. 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.

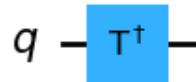


Figure 3.11: T^\dagger Gate

In the operator Notation it is:

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

One might wonder why the T gate is called the $\pi/8$ gate when it is $\pi/4$ that appears in the definition. The reason is that the gate has historically often been referred to as the $\pi/8$ gate simply because up to an unimportant global phase T is

equal to a gate which has $\exp(\pm i\pi/8)$ appearing on its diagonals. 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 $e^{-i\pi/4}$. 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:

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

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.

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

Example 3.3.10. Let x be a real number and A a matrix such that $A^2 = I$. Show that

$$\exp(iAx) = \cos(x)I + i\sin(x)A$$

Let $|v\rangle$ be an eigenvector of A with eigenvalue λ . It then follows that $A^2|v\rangle = \lambda^2|v\rangle$, and furthermore we have that $A^2|v\rangle = I|v\rangle = |v\rangle$ by assumption. We obtain that $\lambda^2 = 1$ and therefore the only possible eigenvalues of A are $\lambda = \pm 1$. Let $|v_1\rangle, \dots, |v_k\rangle$ be the eigenvectors with eigenvalue 1 and $|v_{k+1}\rangle, \dots, |v_n\rangle$ be the eigenvectors with eigenvalue -1 . By the spectral decomposition, we can write:

$$A = \sum_{i=1}^k |v_i\rangle\langle v_i| - \sum_{i=k+1}^n |v_i\rangle\langle v_i|$$

so by the definition of operator functions we have:

$$\exp(\iota Ax) = \sum_{i=1}^k \exp(\iota x) |v_i\rangle \langle v_i| + \sum_{i=k+1}^n \exp(-\iota x) |v_i\rangle \langle v_i|$$

By Euler's identity we have:

$$\exp(\iota Ax) = \sum_{i=1}^k (\cos(x) + \iota \sin(x)) |v_i\rangle \langle v_i| + \sum_{i=k+1}^n (\cos(x) - \iota \sin(x)) |v_i\rangle \langle v_i|$$

Grouping terms, we obtain:

$$\exp(\iota Ax) = \cos(x) \sum_{i=1}^n |v_i\rangle \langle v_i| + \iota \sin(x) \left(\sum_{i=1}^k |v_i\rangle \langle v_i| - \sum_{i=k+1}^n |v_i\rangle \langle v_i| \right)$$

Using the spectral decomposition and definition of I , we therefore obtain the desired relation:

$$\exp(\iota Ax) = \cos(x)I + \iota \sin(x)A$$

Since all of the Pauli matrices satisfy $A^2 = I$ i.e they are self inverse, for $\theta \in \mathbb{R}$, we can apply this obtained relation. This will be useful for application in Hamiltonian simulation.

Example 3.3.11. Functions of the Pauli matrices let \vec{v} be any real, three-dimensional unit vector and θ a real number. Let $f(\cdot)$ be any function from complex numbers to complex numbers. Let \vec{n} be a normalized vector in three dimensions, and let θ be real. Show that

$$f(\theta \vec{n} \cdot \vec{\sigma}) = \frac{f(\theta) + f(-\theta)}{2} I + \frac{f(\theta) - f(-\theta)}{2} \vec{n} \cdot \vec{\sigma}$$

where $\vec{n} \cdot \vec{\sigma} = \sum_{i=1}^3 n_i \cdot \sigma_i$, this implies, $\theta \vec{n} \cdot \vec{\sigma} = \sum_{i=1}^3 \theta n_i \cdot \sigma_i$.

$$\begin{aligned} \sum_{i=1}^3 \theta n_i \sigma_i &= \theta n_1 |+\rangle \langle +| - \theta n_1 |-\rangle \langle -| + \theta n_2 |+\iota\rangle \langle +\iota| - \theta n_2 |-\iota\rangle \langle -\iota| + \theta n_3 |0\rangle \langle 0| - \theta n_3 |1\rangle \langle 1| \\ &= \theta \begin{bmatrix} n_3 & n_1 - \iota n_2 \\ n_1 + \iota n_2 & -n_3 \end{bmatrix} \end{aligned}$$

where \vec{n} is a real, three dimensional unit vector and $\theta \in \mathbb{R}$, $n_1, n_2, n_3 \in \mathbb{R}$. Note that the above given matrix is clearly a Hermitian matrix, thus has a unitary decomposition (has complete set of eigenvectors spanning \mathbb{C}^2). Now, upon solving for the eigen values we get,

$$\begin{vmatrix} n_3 - \lambda & n_1 + \iota n_2 \\ n_1 + \iota n_2 & -n_2 - \lambda \end{vmatrix} = 0$$

this results in the eigenvalues as (using the property of \vec{n} being a unit vector)

$$\lambda = \pm \sqrt{n_1^2 + n_2^2 + n_3^2}$$

$$\lambda = \pm 1$$

Say it's eigen values are $|v_+\rangle$ corresponding to $\lambda = 1$ and $|v_-\rangle$ corresponding to $\lambda = -1$. Thus, we get,

$$\begin{aligned} \vec{n} \cdot \vec{\sigma} &= |v_+\rangle \langle v_+| - |v_-\rangle \langle v_-| \\ \sum_{i=1}^3 \theta n_i \sigma_i &= \theta(|v_+\rangle \langle v_+| - |v_-\rangle \langle v_-|) \end{aligned}$$

Now,

$$f(\theta \vec{n} \cdot \vec{\sigma}) = f(\theta) |v_+\rangle \langle v_+| + f(-\theta) |v_-\rangle \langle v_-|$$

From the completeness relationship, we can write Identity as $I = |v_+\rangle \langle v_+| + |v_-\rangle \langle v_-|$, hence, For a observable on a 2-dimensional Hilbert space with eigenvalues $\lambda = \pm 1$ has projectors

$$\begin{aligned} \frac{I + \vec{n} \cdot \vec{\sigma}}{2} &= |v_+\rangle \langle v_+| \\ \frac{I - \vec{n} \cdot \vec{\sigma}}{2} &= |v_-\rangle \langle v_-| \end{aligned}$$

Substituting this in the equation we get,

$$f(\theta \vec{n} \cdot \vec{\sigma}) = f(\theta) \frac{I + \vec{n} \cdot \vec{\sigma}}{2} + f(-\theta) \frac{I - \vec{n} \cdot \vec{\sigma}}{2}$$

On simplifying this we get,

$$f(\theta \vec{n} \cdot \vec{\sigma}) = \frac{f(\theta) + f(-\theta)}{2} I + \frac{f(\theta) - f(-\theta)}{2} \vec{n} \cdot \vec{\sigma}$$

In case we write the function as $\exp(\iota \theta \vec{n} \cdot \vec{\sigma})$ we get,

$$\exp(\iota \theta \vec{n} \cdot \vec{\sigma}) = \cos(\theta) I + \iota \sin(\theta) \vec{n} \cdot \vec{\sigma}$$

3.3.5 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. The Pauli matrices give rise to three useful classes of Unitary matrices when they are exponentiated, the rotation operators about the x, y, z axes.

R_X Gate

It is a rotation about X axis of Bloch Sphere by an angle θ . Putting $\vec{n} = (1, 0, 0)$ in the previous equation and putting $\theta = -\theta/2$. This is denoted as R_θ^X . The matrix representation of R_θ^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. Note that the value $\pi/2$ given in



Figure 3.12: R_X Gate

the figure 3.12 denotes the angle by which we rotate around the X Axis of Bloch Sphere. Note that $R_\pi^X = -iX$.

R_Y Gate

It is a rotation about Y axis of Bloch Sphere by an angle θ . This is denoted as R_θ^Y . The matrix representation of R_θ^Y is:

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

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. Note here the value $\pi/2$ denotes the



Figure 3.13: R_Y Gate

angle by which we rotate around the Y Axis of Bloch Sphere. Note that $R_\pi^Y = -iY$.

 R_Z Gate

It is a rotation about Z axis of Bloch Sphere by an angle θ . This is denoted as R_θ^Z . The matrix representation of R_θ^Z is:

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

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. Note here the value $\pi/2$ denotes the

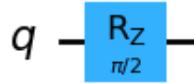


Figure 3.14: R_Z Gate

angle by which we rotate around the Z Axis of Bloch Sphere. Note that $R_\pi^Z = -\iota Z$.

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

R_n Gate

If $\hat{n} = (n_x, n_y, n_z)$ is a real unit vector in three dimensions then we generalize the previous definitions by defining a rotation by θ about the \hat{n} axis by the equation

$$R_{\hat{n}}(\theta) = \exp(-\iota\theta\hat{n} \cdot \vec{\sigma}/2) = \cos\left(\frac{\theta}{2}\right)I - \iota \sin\left(\frac{\theta}{2}\right)(n_xX + n_yY + n_zZ)$$

where $\vec{\sigma}$ denotes the three component vector (X, Y, Z) of Pauli matrices. Thus, in matrix form, we can write $R_n(\theta)$ as:

$$\begin{bmatrix} \cos \frac{\theta}{2} - \iota \sin \frac{\theta}{2} n_z & -\sin \frac{\theta}{2} (n_y + \iota n_x) \\ \sin \frac{\theta}{2} (n_y - \iota n_x) & \cos \frac{\theta}{2} + \iota \sin \frac{\theta}{2} n_z \end{bmatrix}$$

3.3.6 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 θ 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 θ and ϕ 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. Note here the values $\pi/2, \pi/2, \pi/2$



Figure 3.15: U_3 Gate

denotes the angles θ, ϕ, λ by which we rotate around the Bloch Sphere respectively.

Example 3.3.12. Show that, up to a global phase, the $\pi/8$ gate satisfies $T = R_z(\pi/4)$

Recall that the T gate is defined as:

$$T = \begin{bmatrix} 1 & 0 \\ 0 & \exp(i\pi/4) \end{bmatrix}$$

We observe that:

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

Example 3.3.13. Express the Hadamard gate H as a product of R_x and R_z rotations and $e^{i\varphi}$ for some φ .

We claim that $H = R_z(\pi/2)R_x(\pi/2)R_z(\pi/2)$ up to a global phase of $e^{-i\pi/2}$. Doing a computation to verify this claim, we see that:

$$\begin{aligned} R_z(\pi/2)R_x(\pi/2)R_z(\pi/2) &= \begin{bmatrix} e^{-i\pi/4} & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \begin{bmatrix} \cos(\frac{\pi}{4}) & -i\sin(\frac{\pi}{4}) \\ -i\sin(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) \end{bmatrix} \begin{bmatrix} e^{-i\pi/4} & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \\ &= \begin{bmatrix} e^{-i\pi/4} & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{i}{\sqrt{2}} \\ -\frac{i}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} e^{-i\pi/4} & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} e^{-i\pi/4} & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \begin{bmatrix} e^{-i\pi/4} & -ie^{i\pi/4} \\ -ie^{-i\pi/4} & e^{i\pi/4} \end{bmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} e^{-i\pi/2} & -e^{i\pi/2} \\ -e^{-i\pi/2} & e^{i\pi/2} \end{bmatrix} \\ &= \frac{e^{-i\pi/2}}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ &= e^{-i\pi/2} H \end{aligned}$$

Thus, we get,

$$e^{i\pi/2} R_z(\pi/2)R_x(\pi/2)R_z(\pi/2) = H$$

Example 3.3.14. Prove that $(\hat{\mathbf{n}} \cdot \sigma)^2 = I$

$$\begin{aligned} (\hat{\mathbf{n}} \cdot \sigma)^2 &= (n_x X + n_y Y + n_z Z)^2 \\ &= n_x^2 X^2 + n_y^2 Y^2 + n_z^2 Z^2 + n_x n_y (XY + YX) + n_x n_z (XZ + ZX) + n_y n_z (YZ + ZY) \end{aligned}$$

Recall that $\{\sigma_i, \sigma_j\} = 2\delta_{ij}I$, using this, we have that:

$$(\hat{\mathbf{n}} \cdot \sigma)^2 = (n_z^2 + n_y^2 + n_z^2)I = I$$

where we use the fact that $\hat{\mathbf{n}}$ is a vector of unit length.

Example 3.3.15. One reason why the $\hat{R}_n(\theta)$ operators are referred to as rotation operators is the following fact, which you are to prove. Suppose a single qubit has a state represented by the Bloch vector λ . Then, the effects of the rotation $\hat{R}_n(\theta)$ on the state is to rotate by an angle θ about the \hat{n} axis of the Bloch sphere. This fact explains the rather mysterious looking factors of two in the definition of the rotation matrices.

Let λ be an arbitrary Bloch vector. Without the loss of generality, we can express λ in a coordinate system such that \hat{n} is aligned with z axis, so it suffices to consider how the state behaves under the application $R_z(\theta)$. Let $\lambda = ()$ be the vector expressed in this coordinate system. We know that the density operator corresponding to this Bloch vector is given by:

$$\rho = \frac{I + \lambda \cdot \sigma}{2}$$

We now observe how ρ transforms under the conjugation of $R_z(\theta)$.

$$\begin{aligned} R_z(\theta)\rho R_z(\theta)^\dagger &= R_z(\theta)\rho R_z(-\theta) \\ &= R_z(\theta) \left(\frac{I + \lambda_x X + \lambda_y Y + \lambda_z Z}{2} \right) R_z(-\theta) \end{aligned}$$

Using that $XZ = -ZX$, we make the observation that:

$$\begin{aligned} R_z(\theta)X &= \left(\cos\left(\frac{\theta}{2}\right) I - \iota \sin\left(\frac{\theta}{2}\right) Z \right) X \\ &= X \left(\cos\left(\frac{\theta}{2}\right) I + \iota \sin\left(\frac{\theta}{2}\right) Z \right) \\ &= X \left(\cos\left(\frac{-\theta}{2}\right) I - \iota \sin\left(\frac{-\theta}{2}\right) Z \right) \\ &= XR_z(-\theta) \end{aligned}$$

Similarly, we find that $R_z(\theta)Y = YR_z(-\theta)$ (same anticommutation) and that $R_z(\theta)Z = ZR_z(\theta)$ (all terms commute). With this, for expression $R_z(\theta)\rho R_z(\theta)^\dagger$ simplifies to:

$$\begin{aligned} R_z(\theta)\rho R_z(\theta)^\dagger &= R_z(\theta) \left(\frac{I + \lambda_z X + \lambda_y Y + \lambda_z Z}{2} \right) R_z(-\theta) \\ &= \frac{I + \lambda_x X R_z(-2\theta) + \lambda_y Y R_z(-2\theta) + \lambda_z Z}{2} \end{aligned}$$

Calculating each of the terms in the above expression, we have:

$$\begin{aligned}
XR_z(-2\theta) &= X \left(\cos\left(\frac{-2\theta}{2}\right) - i \sin\left(\frac{-2\theta}{2}\right) Z \right) \\
&= X(\cos(\theta) + i \sin(\theta)Z) \\
&= \cos(\theta)X + i \sin(\theta)XZ \\
&= \cos(\theta)X + i \sin(\theta)(-\iota Y) \\
&= \cos(\theta)X + \sin(\theta)Y \\
YR_z(-2\theta) &= Y(\cos(\theta) + i \sin(\theta)Z) \\
&= \cos(\theta)Y + i \sin(\theta)YZ \\
&= \cos(\theta)Y + i \sin(\theta)(\iota X) \\
&= \cos(\theta)Y - \sin(\theta)X
\end{aligned}$$

Plugging these back into the expression for $R_z(\theta)\rho R_z(\theta)^\dagger$ and collecting like terms, we have:

$$R_z(\theta)\rho R_z(\theta)^\dagger = \frac{I + (\lambda_x \cos(\theta) - \lambda_y \sin(\theta))X + (\lambda_x \sin(\theta) + \lambda_y \cos(\theta))Y + \lambda_z Z}{2}$$

From this expression, we can read off the new Bloch vector λ' after conjugation by $R_z(\theta)$ to be:

$$\lambda' = (\lambda_x \cos(\theta) - \lambda_y \sin(\theta), \lambda_x \sin(\theta) + \lambda_y \cos(\theta), \lambda_z)$$

Alternatively, suppose we apply the 3-dimensional rotation matrix $A_z(\theta)$ to the original bloch vector λ . We have that:

$$A_z(\theta)\lambda = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda_x \\ \lambda_y \\ \lambda_z \end{bmatrix} = \begin{bmatrix} \lambda_x \cos \theta - \lambda_y \sin \theta \\ \lambda_x \sin \theta + \lambda_y \cos \theta \\ \lambda_z \end{bmatrix}$$

We see that we end up with the same resulting vector λ' . We conclude that the conjugation of ρ under $R_z(\theta)$ has the equivalent effect to rotating the Bloch vector by θ about the z-axis, and hence the effect of $R_n(\theta)$ on one qubit state is to rotate it by an angle θ about $\hat{\mathbf{n}}$.

Example 3.3.16. Show that $XYX = -Y$ and use this to prove that $XR_y(\theta)X = R_y(-\theta)$.

For the first, claim we use that $XY = -YX$ and $X^2 = I$ to obtain that

$$XYX = -YXX = -YI = -Y$$

Using this, we have that:

$$\begin{aligned}
XR_y(\theta)X &= X \left(\cos\left(\frac{\theta}{2}\right) I - \iota \sin\left(\frac{\theta}{2}\right) Y \right) X \\
&= \cos\left(\frac{\theta}{2}\right) XIX - \iota \sin\left(\frac{\theta}{2}\right) XYX \\
&= \cos\left(\frac{\theta}{2}\right) I + \iota \sin\left(\frac{\theta}{2}\right) Y \\
&= \cos\left(-\frac{\theta}{2}\right) I - \iota \sin\left(-\frac{\theta}{2}\right) Y \\
&= R_y(-\theta)
\end{aligned}$$

Example 3.3.17. An arbitrary single qubit unitary operator can be written in the form

$$U = \exp(\iota\alpha)R_n(\theta)$$

for some real numbers α and θ , and a real three-dimensional unit vector \hat{n} .

1. Prove this fact.
2. Find values for α, θ , and \hat{n} giving the Hadamard gate H .
3. Find values for α, θ and \hat{n} giving the phase gate

$$S = \begin{bmatrix} 1 & 0 \\ 0 & \iota \end{bmatrix}$$

1. By definition, for any unitary operator U we have that $U^\dagger U = I$, so for any state vector $\langle \psi | \psi \rangle = \langle \psi | U^\dagger U | \psi \rangle$. Therefore, all unitary Us are norm-preserving, and hence for a single qubit correspond to some reflection/rotation in 3-dimensional space (up to a global phase factor). Hence, we can write $U = \exp(\iota\alpha)R_n(\theta)$ for some \hat{n} (rotation axis), θ (rotation angle) and α (global phase). Any 2×2 unitary matrix for $a^2 + b^2 + c^2 + d^2 = 1$ can be written as,

$$U = e^{\iota\alpha} \begin{bmatrix} a + \iota b & c + \iota d \\ -c + \iota d & a - \iota b \end{bmatrix}$$

Consider, the given form for U . Note that we can write $R_n(\theta) = e^{-\iota\frac{\theta}{2}(n \cdot \sigma)}$ by substituting the values of X, Y, Z pauli matrices and thus arriving at the following matrix as:

$$U = e^{\iota\alpha} \begin{bmatrix} \cos\frac{\theta}{2} - \iota \sin\frac{\theta}{2} n_z & -\sin\frac{\theta}{2}(n_y + \iota n_x) \\ \sin\frac{\theta}{2}(n_y - \iota n_x) & \cos\frac{\theta}{2} + \iota \sin\frac{\theta}{2} n_z \end{bmatrix}$$

As, $n_x^2 + n_y^2 + n_z^2 = 1$ this has the same form as the general U , also, there are four independent variables- any two of the n_x, n_y, n_z as they are constrained by the equation $n_x^2 + n_y^2 + n_z^2 = 1$ and θ, α . hence any arbitrary 2×2 unitary matrix can be written as $U = e^{i\alpha}R_n(\theta)$.

2. Using the fact that $H = \frac{X+Z}{\sqrt{2}}$, and that modulo factor of i that X/Z correspond to rotations $R_x(\pi)$ and $R_z(\pi)$, we find that:

$$\begin{aligned} H &= \frac{iR_x(\pi) + iR_z(\pi)}{\sqrt{2}} = i \left(\frac{2 \cos \frac{\pi}{2} I - i \sin \frac{\pi}{2} X - i \sin \frac{\pi}{2} Z}{\sqrt{2}} \right) \\ &= e^{i\frac{\pi}{2}} \left(\cos \left(\frac{\pi}{2} \right) I - i \sin \left(\frac{\pi}{2} \right) \left(\frac{1}{\sqrt{2}} X + 0Y + \frac{1}{\sqrt{2}} Z \right) \right) \end{aligned}$$

Note that in the second equality we use that $\cos \left(\frac{\pi}{2} \right) = 0$ and hence $\frac{2}{\sqrt{2}} \cos \left(\frac{\pi}{2} \right) = \cos \left(\frac{\pi}{2} \right)$. From the last expression, we can read off using the definition of $R_n(\theta)$ that $n = \left(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}} \right)$, $\theta = \pi$, and $\alpha = \frac{\pi}{2}$.

3. We observe that:

$$R_z \left(\frac{\pi}{2} \right) = \begin{bmatrix} e^{-i\pi/4} & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} = e^{-i\pi/4} \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

Hence:

$$S = e^{i\pi/4} R_z \left(\frac{\pi}{2} \right)$$

from which we obtain that $\hat{n} = \hat{z} = (0, 0, 1)$, $\theta = \frac{\pi}{2}$, and $\alpha = \frac{\pi}{4}$.

Note that one can just use the definition:

$$R_n(\theta) = \exp(-i\theta \hat{n} \cdot \sigma/2) = \cos \left(\frac{\theta}{2} \right) I - i \sin \left(\frac{\theta}{2} \right) (n_x X + n_y Y + n_z Z)$$

and the fact that $H = (X + Z)/\sqrt{2}$, to arrive at $\cos \left(\frac{\theta}{2} \right) = 0$, $n_x = n_z = \frac{1}{\sqrt{2}}$, $n_y = 0$.

An arbitrary unitary operator on a single qubit can be written in many ways as a combination of rotations, together with global phase shifts on the qubit. The following theorem provides a means of expressing an arbitrary single qubit rotation that will be particularly useful in later applications to controlled operations.

Theorem 3.3.1. (*Z-Y decomposition for a single qubit*) Suppose U is a unitary operation on a single qubit. Then there exist a real number α, β, γ and δ such that

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta)$$

Proof. Since U is unitary, the rows and columns of U are orthonormal, from which it follows that there exist real numbers α, β, γ , and δ such that

$$U = \begin{bmatrix} e^{\iota(\alpha-\beta/2-\delta/2)} \cos \frac{\gamma}{2} & -e^{\iota(\alpha-\beta/2+\delta/2)} \sin \frac{\gamma}{2} \\ e^{\iota(\alpha+\beta/2-\delta/2)} \sin \frac{\gamma}{2} & e^{\iota(\alpha+\beta/2+\delta/2)} \cos \frac{\gamma}{2} \end{bmatrix}$$

Thus, the equation now follows immediately from the definition of the rotation matrices and matrix multiplication. \square

Example 3.3.18. Explain why any single qubit unitary operator may be written in the previous form.

Let U be a single qubit unitary operator. we then have that $U^\dagger U = I$, so identifying:

$$U = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = [\mathbf{v}_1 \ \mathbf{v}_2]$$

we obtain that:

$$\begin{bmatrix} |a|^2 + |c|^2 & a^*b + c^*d \\ ab^* + cd^* & |b|^2 + |d|^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

From the diagonal entries we obtain that $|a|^2 + |c|^2 = 1$ and $|b|^2 + |d|^2 = 1$ and from the off-diagonal entries we obtain that $\langle v_1, v_2 \rangle = 0$, and hence the columns of U are orthonormal. From the fact that $|v_1|$ is normalized, we can parametrize the magnitude of the entries with $\gamma \in \mathbb{R}$ such that:

$$|a| = \cos \frac{\gamma}{2}, \quad |c| = \sin \frac{\gamma}{2}$$

from the orthogonality $U^\dagger = U^{-1}$ we get,

$$U^{-1} = \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = U^\dagger = \begin{bmatrix} a^* & c^* \\ b^* & d^* \end{bmatrix}$$

we further obtain that $b = -c^*$ and $d = a^*$, from which we have that $|b| = |c|$ and $|d| = |a|$. Thus, we have the following matrix:

$$U = \begin{bmatrix} a & -c^* \\ c & a^* \end{bmatrix}$$

Furthermore, (also from orthogonality) we can parametrize such that for any $\arg(a) = -\frac{\beta}{2} - \frac{\delta}{2}$ and $\arg(b) = \frac{\beta}{2} - \frac{\delta}{2}$. For $\beta, \delta \in \mathbb{R}$. Finally, multiplying U by a complex phase

$e^{\iota\alpha}$ for $\alpha \in \mathbb{R}$ preserves the unitarity of U and the orthonormality of the columns. Combining these facts gives the form as desired. We can write,

$$\begin{aligned} U &= \begin{bmatrix} e^{\iota(\alpha-\beta/2-\delta/2)} \cos \frac{\gamma}{2} & -e^{\iota(\alpha-\beta/2+\delta/2)} \sin \frac{\gamma}{2} \\ e^{\iota(\alpha+\beta/2-\delta/2)} \sin \frac{\gamma}{2} & e^{\iota(\alpha+\beta/2+\delta/2)} \cos \frac{\gamma}{2} \end{bmatrix} = e^{\iota\alpha} \begin{bmatrix} e^{\iota(-\beta/2-\delta/2)} \cos \frac{\gamma}{2} & -e^{\iota(-\beta/2+\delta/2)} \sin \frac{\gamma}{2} \\ e^{\iota(\beta/2-\delta/2)} \sin \frac{\gamma}{2} & e^{\iota(\beta/2+\delta/2)} \cos \frac{\gamma}{2} \end{bmatrix} \\ &= e^{\iota\alpha} \begin{bmatrix} a & -c^* \\ c & a^* \end{bmatrix} \end{aligned}$$

which is the general form of a 2×2 unitary matrix as $\det(U) = \cos^2 \frac{\gamma}{2} + \sin^2 \frac{\gamma}{2} = 1$.

Example 3.3.19. (X-Y decomposition of rotations) Give a decomposition analogous to the previous theorem but using R_x instead of R_z .

We simply use the fact that we can do a change of reference, to interchange the x and z axes. We know that the hadamard H is unitary and self-inverse implements the appropriate swap between x and z axes, while send y to $-y$. So instead of working with U directly, consider

$$U' = HUH^{-1} = HUH$$

which is the frame-rotated U . Now, we can use the following ZY decomposition of U' .

$$U' = e^{\iota\alpha} R_z(\beta) R_y(\gamma) R_z(\delta)$$

What does this imply for U ? We will see that the above implies an XY-decomposition for U .

$$\begin{aligned} U &= HU'H = e^{\iota\alpha} (HR_z(\beta)H)(HR_y(\gamma)H)(HR_z(\delta)H) \\ &= e^{\iota\alpha} R_x(\beta) R_y(-\gamma) R_x(\delta) \end{aligned}$$

Example 3.3.20. Suppose \hat{m} and \hat{n} are non-parallel real unit vectors in three dimensions. Use the theorem to show that an arbitrary single qubit unitary U may be written

$$U = e^{\iota\alpha} R_{\hat{n}}(\beta) R_{\hat{m}}(\gamma) R_{\hat{n}}(\delta)$$

for appropriate choices of α, β, γ and δ . The same arguments in above example could be generalize. We could choose any two orthogonal axes \hat{n}, \hat{m} for rotations in our decomposition, using the fact that we can always frame rotate to z and x axes.

Consider the following corollary, which is the key to the construction of controlled multi-qubit unitary operations, as explained.

Corollary 3.3.2. Suppose U is a unitary gate on a single qubit. Then there exist unitary operators A, B, C on a single qubit such that $ABC = I$ and $U = e^{i\alpha}AXBXC$, where α is some overall phase factor.

Proof. Set $A = R_z(\beta)R_y(\gamma/2)$, $B = R_y(-\gamma/2)R_z(-(\delta+\beta)/2)$ and $C = R_z((\delta-\beta)/2)$. Note that

$$ABC = R_z(\beta)R_y\left(\frac{\gamma}{2}\right)R_y\left(-\frac{\gamma}{2}\right)R_z\left(-\frac{\delta+\beta}{2}\right)R_z\left(\frac{\delta-\beta}{2}\right)$$

Since $X^2 = I$ and $XR_Y(\theta)X = R_y(-\theta)$, we see that,

$$AXBX = XR_y\left(-\frac{\gamma}{2}\right)XXR_z\left(-\frac{\delta+\beta}{2}\right)X = R_y\left(\frac{\gamma}{2}\right)R_z\left(\frac{\delta+\beta}{2}\right)$$

Thus

$$\begin{aligned} AXBXC &= R_z(\beta)R_y\left(\frac{\gamma}{2}\right)R_y\left(\frac{\gamma}{2}\right)R_z\left(\frac{\delta+\beta}{2}\right)R_z\left(\frac{\delta-\beta}{2}\right) \\ &= R_z(\beta)R_y(\gamma)R_z(\delta) \end{aligned}$$

Thus, $U = e^{i\alpha}AXBXC$ and $ABC = I$, as required. \square

Example 3.3.21. Give A, B, C and α for the Hadamard Gate.

From the proof recall that U is unitary such that

$$U = \begin{bmatrix} e^{i(\alpha-\beta/2-\delta/2)} \cos \frac{\gamma}{2} & -e^{i(\alpha-\beta/2+\delta/2)} \sin \frac{\gamma}{2} \\ e^{i(\alpha+\beta/2-\delta/2)} \sin \frac{\gamma}{2} & e^{i(\alpha+\beta/2+\delta/2)} \cos \frac{\gamma}{2} \end{bmatrix}$$

This matrix becomes H when $\alpha = \gamma = \frac{\pi}{2}, \delta = \pi, \beta = 0$. Thus, from the previous corollary, we see that $A = R_z(\beta)R_y(\gamma/2) = R_y\left(\frac{\pi}{4}\right)$, $B = R_y(-\gamma/2)R_z(-(\delta+\beta)/2) = R_y\left(-\frac{\pi}{4}\right)R_z\left(-\frac{\pi}{2}\right)$, $C = R_z((\delta-\beta)/2) = R_z\left(\frac{\pi}{2}\right)$ and $\alpha = 0$.

Example 3.3.22. (Circuit identities) It is useful to be able to simplify the circuits by inspection, using well-known identities. Prove the following three identities.

$$HXH = Z; \quad HYH = -Y; \quad HZH = X$$

Clearly,

$$HXH = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix} = Z$$

$$\begin{aligned} HYH &= \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & 2i \\ -2i & 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & 2i \\ -2i & 0 \end{bmatrix} = -Y \\ HZH &= HHXHH = X \end{aligned}$$

Example 3.3.23. Use the previous example to show that $HTH = R_x(\pi/4)$, up to a global phase.

Recall that $T = R_z(\pi/4)$ up to a global phase. Thus, $HTH = HR_z(\pi/4)H$. Thus,

$$HTH = HR_z\left(\frac{\pi}{4}\right)H = H\left(\cos\frac{\pi}{8}I - \iota\sin\frac{\pi}{8}Z\right)H = \cos\frac{\pi}{8}I - \iota\sin\frac{\pi}{8}X = R_x\left(\frac{\pi}{4}\right)$$

Example 3.3.24. (Composition of single qubit operations) The Bloch representation gives a nice way to visualize the effect of composing two rotations:

1. Prove that if a rotation through an angle β_1 about the axis \hat{n}_1 is followed by a rotation through an angle β_2 about an axis \hat{n}_2 , then the overall rotation is through an angle β_{12} about an axis \hat{n}_{12} given by

$$\begin{aligned} c_{12} &= c_1c_2 - s_1s_2\hat{n}_1 \cdot \hat{n}_2 \\ s_{12}\hat{n}_{12} &= s_1c_2\hat{n}_1 + c_1s_2\hat{n}_2 + s_1s_2\hat{n}_2 \times \hat{n}_1 \end{aligned}$$

2. Show that if $\beta_1 = \beta_2$ and $\hat{n}_1 = \hat{z}$ these equations simplify to

$$\begin{aligned} c_{12} &= c^2 - s^2\hat{z} \cdot \hat{n}_2 \\ s_{12}\hat{n}_{12} &= sc(\hat{z} + \hat{n}_2) - s^2\hat{n}_2 \times \hat{z} \end{aligned}$$

where $c = c_1$ and $s = s_1$.

We are required to compute $R_{\hat{n}_2}(\beta_2)R_{\hat{n}_1}(\beta_1)$

$$\begin{aligned} R_{\hat{n}_2}(\beta_2)R_{\hat{n}_1}(\beta_1) &= \left(\cos\frac{\beta_2}{2}I - \iota\sin\frac{\beta_2}{2}\hat{n}_2 \cdot \vec{\sigma}\right)\left(\cos\frac{\beta_1}{2}I - \iota\sin\frac{\beta_1}{2}\hat{n}_1 \cdot \vec{\sigma}\right) \\ &= \cos\frac{\beta_2}{2}\cos\frac{\beta_1}{2} - \iota\cos\frac{\beta_2}{2}\sin\frac{\beta_1}{2}\hat{n}_1 \cdot \vec{\sigma} - \iota\sin\frac{\beta_2}{2}\cos\frac{\beta_1}{2}\hat{n}_2 \cdot \vec{\sigma} \\ &\quad - \sin\frac{\beta_2}{2}\sin\frac{\beta_1}{2}(\hat{n}_2 \cdot \vec{\sigma})(\hat{n}_1 \cdot \vec{\sigma}) \\ &= [c_2c_1 - s_2s_1(\hat{n}_2 \cdot \hat{n}_1)]I - \iota[c_1s_2\hat{n}_2 + c_2s_1\hat{n}_1 + s_2s_1(\hat{n}_2 \times \hat{n}_1)] \cdot \vec{\sigma} \end{aligned}$$

The proof of the identity $(\hat{n}_2 \cdot \vec{\sigma})(\hat{n}_1 \cdot \vec{\sigma}) = (\hat{n}_2 \cdot \hat{n}_1)I + \iota(\hat{n}_2 \times \hat{n}_1) \cdot \vec{\sigma}$ is as follows: Recall the familiar Pauli matrix relation already proved in the examples

$$\sigma_i\sigma_j = \delta_{ij}I + \iota\sum_{k=1}^3\epsilon_{ijk}\sigma_k$$

Thus, we can then arrive at the following equation

$$a_i \sigma_i b_j \sigma_j = a_i b_j \delta_{ij} + \iota \sum_{k=1}^3 (a_i b_j \epsilon_{ijk}) \sigma_k$$

for $\epsilon_{ijk} = 0$ except for $\epsilon_{123} = \epsilon_{231} = \epsilon_{312} = 1$, and $\epsilon_{321} = \epsilon_{213} = \epsilon_{132} = -1$. Thus, summing over $i, j = 1, 2, 3$ we get,

$$\begin{aligned} \sum_{i=1}^3 \sum_{j=1}^3 a_i \sigma_i b_j \sigma_j &= \sum_{i=1}^3 \sum_{j=1}^3 a_i b_j \delta_{ij} + \iota \sum_{i=1}^3 \sum_{j=1}^3 \sum_{k=1}^3 (a_i b_j \epsilon_{ijk}) \sigma_k \\ (\hat{a} \cdot \vec{\sigma})(\hat{b} \cdot \vec{\sigma}) &= (\hat{a} \cdot \hat{b}) I + \iota \sum_{k=1}^3 (\hat{a} \times \hat{b})_k \sigma_k \end{aligned}$$

where $\hat{a} = (a_1, a_2, a_3)$ and $\hat{b} = (b_1, b_2, b_3)$ and $\vec{\sigma} = (X, Y, Z)$. Thus, in the matrix form, we have the following result:

$$(\hat{a} \cdot \vec{\sigma})(\hat{b} \cdot \vec{\sigma}) = (\hat{a} \cdot \hat{b}) I + \iota (\hat{a} \times \hat{b}) \cdot \vec{\sigma}$$

Identifying this operation to a single rotation $R_{\hat{n}_{12}} = c_{12} I - \iota s_{12} \hat{n}_{12} \cdot \vec{\sigma}$, we arrive at the required relations

$$\begin{aligned} c_{12} &= c_1 c_2 - s_1 s_2 \hat{n}_1 \cdot \hat{n}_2 \\ s_{12} \hat{n}_{12} &= s_1 c_2 \hat{n}_1 + c_1 s_2 \hat{n}_2 + s_2 s_2 \hat{n}_2 \times \hat{n}_1 \end{aligned}$$

Setting $\beta_2 = \beta_2$ and $\hat{n}_1 = \hat{z}$ in the formulas proven above combined with the fact that $c = c_1 = \cos(\beta - 1/2) = \cos(\beta_2/2) = c_2$ (and similarly ($s_1 = s_2 = s$) we have:

$$\begin{aligned} c_{12} &= c^2 - s^2 \hat{z} \cdot \hat{n}_2 \\ s_{12} \hat{n}_{12} &= sc \hat{z} + cs \hat{n}_2 - s^2 \hat{n}_2 \times \hat{z} = sc(\hat{z} + \hat{n}_2) - s^2 \hat{n}_2 \times \hat{z} \end{aligned}$$

3.4 Multi-Qubit Gates

If A is true, then do B . This type of controlled operation is one of the most useful in computing, both classical and quantum. We will now see how complex controlled operations are implemented using quantum circuits built from elementary operations. Quantum Pauli gates and hadamard gate are Unitary and Hermitian, and all the other single qubit gate are Unitary, hence, reversible. In general, we can generalize 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.

3.4.1 Two-qubit Gates

CNOT/CX Gate

It is a two qubit gate, also called as the Controlled-NOT gate. It has two input qubits, known as the control qubit and the target qubit. 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. For the table in 3.11, the 1st

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

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. 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:

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

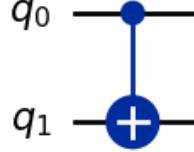


Figure 3.16: CNOT Gate

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:

$$CX |x\rangle |y\rangle = |x\rangle |x \oplus y\rangle$$

where \oplus denotes the XOR operation. If the control qubit is set to $|1\rangle$ then the target qubit is flipped, otherwise the target qubit is left alone. It is a non-linear operation and 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. For the table in 3.12, the 1st Qubit is control

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

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. 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 π 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:

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

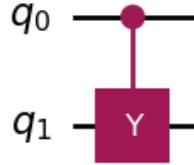


Figure 3.17: CY Gate

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

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

In Outer product representation, it can be written as:

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

In Compact form we can write the CY gate as:

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

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).

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

The truth table is as in table 3.13. 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. In this figure 3.18, the first qubit is

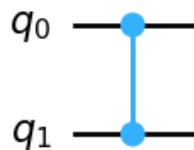


Figure 3.18: CZ Gate

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 π 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. 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')

```

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

```

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. In this figure 3.19, the first qubit

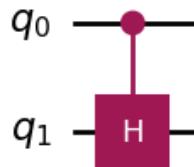


Figure 3.19: CH Gate

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 π 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 2×2 . 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 6×6 and Z is the Pauli-Z gate matrix of size 2×2 . We will later on use this concept to create the CCNOT/Toffoli gate.

More generally, suppose U is an arbitrary single qubit unitary operation. A controlled- U operation is a two qubit operation, again with a control and a target qubit. If the control qubit is set then U is applied to the target qubit, otherwise the target qubit is left alone; that is, $|c\rangle|t\rangle \rightarrow |c\rangle U^c|t\rangle$. The controlled- U operation is represented by the circuit shown in figure 3.20.

Example 3.4.1. (Matrix representation of multi-qubit gates) What is the 4×4 unitary matrix for the circuit. in the computational basis? What is the unitary matrix for the circuit in the computational basis? Depending upon how the input bits are written as x_1x_2 or x_2x_1 , we can write the matrix in different ways. Say as given in the question we first write x_1 and then x_2 then the following is the answer:

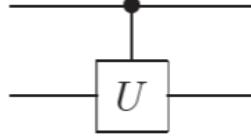


Figure 3.20: Controlled- U operation. The top line is the control qubit, and the bottom line is the target qubit. If the control qubit is set then U is applied to the target, otherwise it is left alone.



x_1

The unitary matrix for the first circuit is given by:

$$I_1 \otimes H_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

The unitary matrix for the second circuit is given by:

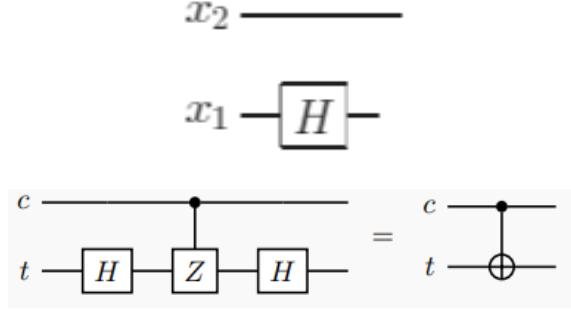
$$H_1 \otimes I_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & -1 \end{bmatrix}$$

Example 3.4.2. (Building a CNOT from controlled- Z gates) Construct a CNOT gate from one controlled- Z gate, that is, the gate whose action in the computational basis is specified by the unitary matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

and two Hadamard gates, specifying the control and target qubits.

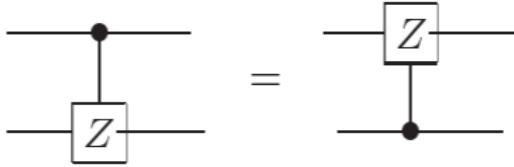
Recall that, $HZH = X$. Hence, to obtain a CNOT gate from a single controlled Z gate, we can conjugate the target qubit with Hadamard gates: we can verify this



via matrix multiplication, using the result from the previous example:

$$\begin{aligned}
 (I_1 \otimes H_2)(CZ_{12})(I_1 \otimes H_2) &= \frac{1}{2} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \\
 &= \frac{1}{2} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 \end{bmatrix} \\
 &= CX_{12}
 \end{aligned}$$

Example 3.4.3. Show that



Recall that, we can write a controlled- U operation as $CZ|b_1\rangle|b_2\rangle = |b_1\rangle\otimes Z^{b_1}|b_2\rangle = (-1)^{b_1\cdot b_2}|b_1\rangle|b_2\rangle$ for computational basis states $b_1, b_2 \in \{0, 1\}$.

Using this form we can write:

$$\begin{aligned}
 CZ_{12}|b_1\rangle|b_2\rangle &= (-1)^{b_1\cdot b_2}|b_1\rangle|b_2\rangle \\
 &= (-1)^{b_1\cdot b_2}|b_1\rangle|b_2\rangle \\
 &= Z^{b_2}|b_1\rangle\otimes|b_2\rangle \\
 &= CZ_{21}|b_1\rangle|b_2\rangle
 \end{aligned}$$

We can also verify this in other ways such as by seeing the action of the controlled gate on all the possible input qubits will be the same.

Example 3.4.4. (CNOT action on density matrices) The CNOT gate is a simple permutation whose action on a density matrix ρ is to rearrange the elements in the matrix. Write out this action explicitly in the computational basis.

Let ρ be an arbitrary density matrix corresponding to a 2 qubit state. In the computational basis, we can write ρ as:

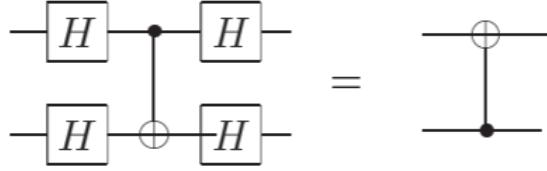
$$\rho = \begin{bmatrix} a_{11} & a_{12} & a_{12} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Studying the action of the CNOT gate on this density matrix, we calculate:

$$\begin{aligned} CX_{12}\rho CX_{12} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} a_{11} & a_{12} & a_{14} & a_{13} \\ a_{21} & a_{22} & a_{24} & a_{23} \\ a_{41} & a_{42} & a_{44} & a_{43} \\ a_{31} & a_{32} & a_{34} & a_{33} \end{bmatrix} \end{aligned}$$

Thus, the action of CNOT on density matrices is to rearrange the elements in the matrix as can be seen from the result.

Example 3.4.5. (CNOT basis transformations) Unlike ideal classical gates ideal quantum gates do not have (as electrical engineers say) ‘high impedance’ inputs. In fact, the role of ‘control’ and ‘target’ are arbitrary -they depend on what basis you think of a devices as operating in. We have described how CNOT behaves with respect to the computational basis, and in this description the state of the control qubit is not changed. However, if we work in a different basis then the control qubit does change: we will show that its phase is flipped depending on the state of the target qubit! Show that using the identity from the previous example that



$CX = (I_1 \otimes H_2)(CZ_{12})(I_1 \otimes H_2)$, and $CZ_{12} = CZ_{21}$, we get,

$$\begin{aligned}
 (H \otimes H)(CX_{12})(H \otimes H) &= (H \otimes H)(I \otimes H)(CZ_{12})(I \otimes H)(H \otimes H) \\
 &= (H \otimes I)(CZ_{12})(H \otimes I) \\
 &= (H \otimes I)CZ_{21}(H \otimes I) \\
 &= CX_{21}
 \end{aligned}$$

which proves the circuit identity. We know already that $CX |c\rangle |t\rangle = |c\rangle |c \oplus t\rangle$. So using the proven identity and the fact that $H|0\rangle = |+\rangle$, $H|1\rangle = |-\rangle$, we obtain the required map. Introducing basis states $|\pm\rangle \equiv (|0\rangle \pm |1\rangle)/\sqrt{2}$, use this circuit identity to show that the effect of a CNOT with the first qubit as control and the second qubit as target is as follows:

$$\begin{aligned}
 |+\rangle |+> &\rightarrow |+\rangle |+> \\
 |-\rangle |+> &\rightarrow |-\rangle |+> \\
 |+\rangle |-> &\rightarrow |-\rangle |-> \\
 |-\rangle |-> &\rightarrow |+\rangle |->
 \end{aligned}$$

Thus, with respect to this new basis, the state of the target qubit is not changed, while the state of the control qubit is flipped if the target starts as *ket-*, otherwise it is left alone. That is, in this basis, the target and control have essentially interchanged roles.

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. 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')

```

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

```

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 3.21.

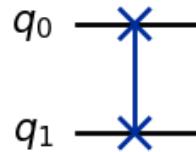


Figure 3.21: SWAP Gate

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: Or using CNOT

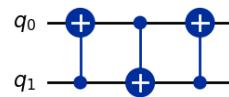


Figure 3.22: SWAP Gate using CNOT Gates

gates as follows:

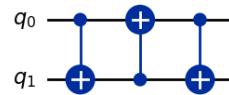


Figure 3.23: SWAP Gate using CNOT Gates

Important Note

The global phase does not matter provided we are doing a measurement at the end (or in between). If we are not doing any measurement then the global phase does matter in the computation.

3.4.2 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. For

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

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 3.24. In this figure 3.24, 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:

$$\begin{aligned}
 CCX |000\rangle &= |000\rangle \\
 CCX |001\rangle &= |001\rangle \\
 CCX |010\rangle &= |010\rangle \\
 CCX |011\rangle &= |011\rangle \\
 CCX |100\rangle &= |100\rangle \\
 CCX |101\rangle &= |101\rangle \\
 CCX |110\rangle &= |111\rangle \\
 CCX |111\rangle &= |110\rangle
 \end{aligned}$$

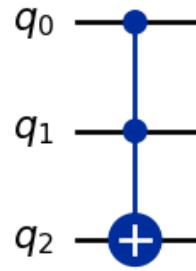


Figure 3.24: Toffoli Gate

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:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \\ \epsilon \\ \zeta \\ \eta \\ \theta \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \\ \epsilon \\ \zeta \\ \eta \\ \theta \end{bmatrix} = \alpha |000\rangle + \beta |001\rangle + \dots + \zeta |101\rangle + \theta |110\rangle + \eta |111\rangle$$

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. For the

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

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')
```

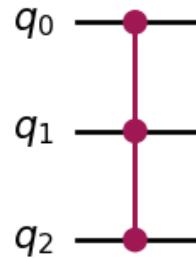


Figure 3.25: Fredkin Gate

The circuit symbol is as shown in figure 3.25. In this figure 3.25, 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 3.26.

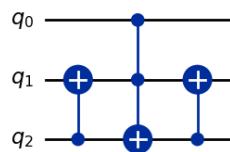


Figure 3.26: Fredkin Gate using CNOT and Toffoli Gates

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 2×2 matrix and $|0\rangle$ and $|1\rangle$ are 2×1 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.

3.5 Implementation of Controlled- U

Now we are required to understand how to implement the controlled- U operation for arbitrary single qubit U , using only single qubit operations and the CNOT gate. Our strategy is a two-part procedure based upon the decomposition $e^{i\alpha}AXBXC$.

The first step will be to apply the phase shift $\exp(i\alpha)$ on the target qubit, controlled by the control qubit. that is, if the control qubit is $|0\rangle$, then the target qubit is left alone, while if the control qubit is $|1\rangle$, a phase shift $e^{i\alpha}$ is applied to the target. A circuit implementing this operation using just a single qubit unitary gate is depicted as in the figure 3.27. to verify this circuit works correctly, note that the effect of the circuit on the right hand side is

$$|00\rangle \rightarrow |00\rangle, \quad |01\rangle \rightarrow |01\rangle, \quad |10\rangle \rightarrow e^{i\alpha}|10\rangle, \quad |11\rangle \rightarrow e^{i\alpha}|11\rangle$$

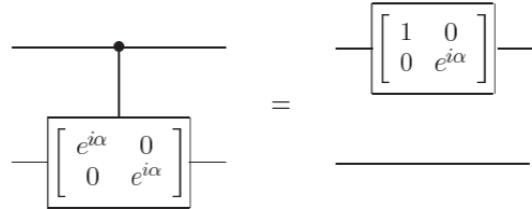


Figure 3.27: Controlled phase shift gate and an equivalent circuit for two qubits

We can now complete the construction of the controlled- U operation, as shown in figure 3.28. to understand why this circuit works, recall that U may be written in the form $U = e^{i\alpha}AXBXC$, where A, B and C are single qubit operations such that $ABC = I$. Suppose that the control qubit is set. then the operation $e^{i\alpha}AXBXC = U$ is applied to the second qubit. If, on the other hand, the control qubit is not set, then the operation $ABC = I$ is applied to the second qubit; that is, no change is made. That is, the circuit implements the controlled- U operation. now that we know how the condition on a single qubit set, what about conditioning on multiple qubits? We've already met one example of multiple qubit conditioning, the Toffoli gate, which flips the third qubit, the target qubit, conditioned on the first two qubits, the control qubits, being set to one. More generally, suppose we have $n + k$ qubits, and U is a k qubit unitary operator. Then we define the controlled operation $C^n(U)$ by the equation.

$$C^n(U) |x_1 x_2 \dots x_n\rangle |\psi\rangle = |x_1 x_2 \dots x_n\rangle |\psi\rangle$$

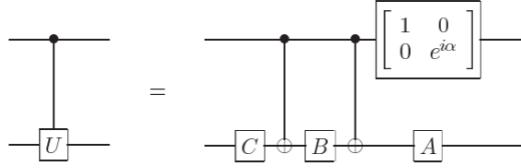


Figure 3.28: Circuit implementing the controlled-U operation for single qubit U . α, A, B and C satisfy $U = \exp(i\alpha)AXBXC, ABC = I$

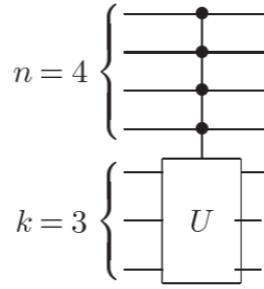


Figure 3.29: Sample circuit representation for the $C^n(U)$ operation, where U is a unitary operator on k qubits, for $n = 4$ and $k = 3$.

where $x_1x_2, \dots x_n$ in the exponent of U means the product of the bits x_1, x_2, \dots, x_n . That is, the operator U is applied to the last k qubits if the first n qubits are equal to one, and otherwise, nothing is done. Such conditional operations are so useful that we introduce a special circuit notation for them, illustrated in figure 3.29. For the following we assume that $k = 1$, for simplicity. larger k can be dealt with using essentially the same methods, however for $k \geq 2$ there is the added complication that we don't know how to perform arbitrary operations on k qubits. Suppose U is a single qubit unitary operator, and V is a unitary operator chosen so that $V^2 = U$. Then the operation $C^2(U)$ may be implemented using the circuit shown in figure 3.30.

Example 3.5.1. Verify that figure 3.30 implements the $C^2(U)$ operation.

Considering all the possible inputs (in the computational basis) let us see the

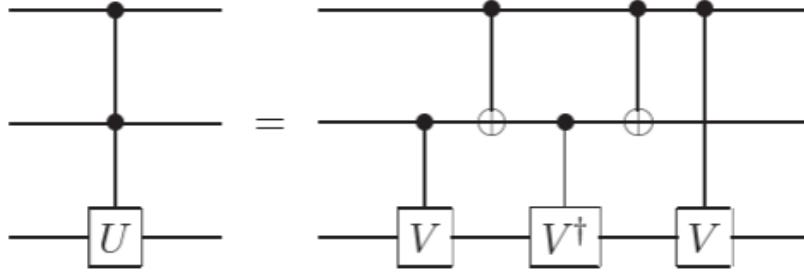


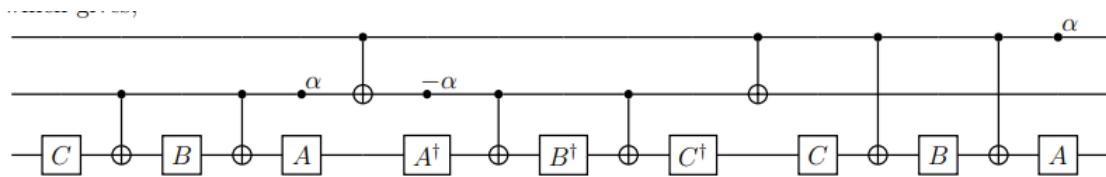
Figure 3.30: Circuit for the $C^2(U)$ gate. V is any unitary operator satisfying $V^2 = U$. The special case $V \equiv (1 - \iota)(I + \iota X)/2$ corresponds to the Toffoli gate.

outcome for both the circuits.

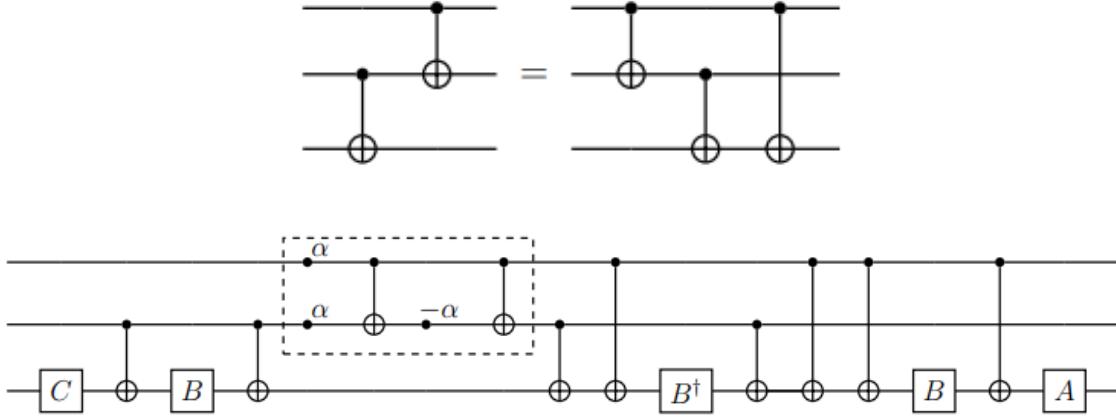
$$\begin{aligned}
 |00\psi\rangle &\rightarrow |00\psi\rangle \rightarrow |00\psi\rangle \rightarrow |00\psi\rangle \rightarrow |00\psi\rangle \rightarrow |00\psi\rangle \\
 |01\psi\rangle &\rightarrow |01(V\psi)\rangle \rightarrow |01(V\psi)\rangle \rightarrow |01(V^\dagger V\psi)\rangle \rightarrow |01(V^\dagger V\psi)\rangle \rightarrow |01(V^\dagger V\psi)\rangle = |01\psi\rangle \\
 |10\psi\rangle &\rightarrow |10\psi\rangle \rightarrow |11\psi\rangle \rightarrow |11(V^\dagger\psi)\rangle \rightarrow |10(V^\dagger\psi)\rangle \rightarrow |10(VV^\dagger\psi)\rangle = |10\psi\rangle \\
 |11\psi\rangle &\rightarrow |11(V\psi)\rangle \rightarrow |10(V\psi)\rangle \rightarrow |10(V\psi)\rangle \rightarrow |11(V\psi)\rangle \rightarrow |11(V^2\psi)\rangle = |11(U\psi)\rangle
 \end{aligned}$$

Hence, the circuit does perform the $C^2(U)$ operation.

Example 3.5.2. Prove that a $C^2(U)$ gate (for any single qubit unitary U) can be constructed using at most eight one-qubit gates, and six controlled-NOTs. Firstly, we apply the circuit if figure 3.29 to the circuit in figure 3.30 which gives, We move



the 6th CNOT left from the 4th one, which involves adding CNOTs after the 4th and 5th CNOTs from first to third qubit, which is due to, and can be checked by consider all the possible inputs. We also see that $AA^\dagger = CC^\dagger = I$. Hence, we get the following circuit. The dashed is diagonal hence commutes with the rest of the components therefore can be moved to the end for convenience. CNOTs 2 and 3, 6 and 7 cancel each other, hence we are left with the circuit This has 8 single qubit gates and 6 CNOTs as desired. The circuit performs the operation, $AXBXB^\dagger XBX C = (VC^\dagger)B^\dagger(A^\dagger V) = V^2 = U$. Therefore, the circuit performs the $C^2(U)$ operation.



Example 3.5.3. Construct a $C^1(U)$ gate for $U = R_x(\theta)$ and $U = R_y(\theta)$, using only CNOT and single qubit gates. Can you reduce the number of single qubit gates need in the construction from three to two?

The familiar Toffoli gate is an especially important special case of $C^2(U)$ operation, the case $C^2(X)$. Defining $V(1 - \iota)(I_\iota X)/2$ and noting that $V^2 = X$, we see that figure 3.31 gives an implementation of the Toffoli gate in terms of one and two qubit operations. From a classical viewpoint this is a remarkable result. Recall that one and two bit classical reversible gates are not sufficient to implement the Toffoli gate, or more generally, universal computation. By contrast, in the quantum case we see that one and two qubit reversible gates are sufficient to implement the Toffoli gate, and will eventually prove that they suffice for universal computation. Ultimately we will show that any unitary operation can be composed into an arbitrary good approximation from just the Hadamard, phase, controlled-NOT and $\pi/8$ gates. Because of the great usefulness of the Toffoli gate it is interesting to see how it can be built from just this gate set. Figure 3.32 illustrates a simple circuit for the Toffoli gate made up of just Hadamard, phase, controlled-NOT and $\pi/8$ gates.

How may we implement $C^n(U)$ gates using our existing repertoire of gates where U is an arbitrary single qubit unitary operation? A particularly simple circuit for achieving this task is illustrated in figure 3.33. The circuit divides up into three stages and makes use of a small number ($n - 1$) working qubits, which all start and end in the state $|0\rangle$. Suppose the control qubits are in the computational basis state $|c_1, c_2, \dots, c_n\rangle$. The first stage of the circuit is to reversible AND all the control bits c_1, \dots, c_n together to produce the product $c_1 \cdot c_2 \dots c_n$. To do this, the first gate in the circuit ANDs c_1 and c_2 together, using a Toffoli gate, changing the state of the first work qubit to $|c_1 \cdot c_2\rangle$. The next Toffoli gate ANDs c_3 with the

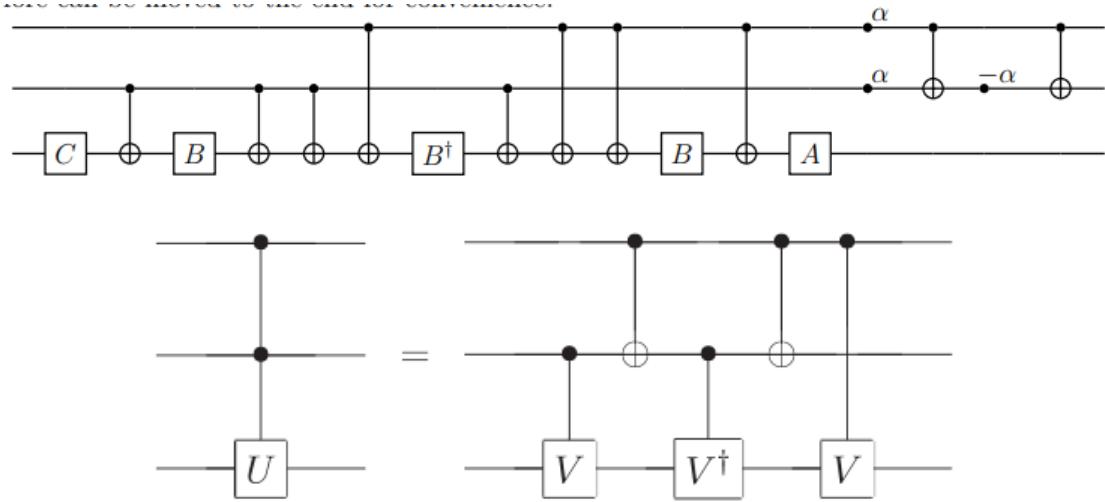


Figure 3.31: Circuit for the $C^2(U)$ gate. V is any unitary operator satisfying $V^2 = U$. The special case $V = (1 - \iota)(I + \iota X)/2$ corresponds to the Toffoli gate.

product $c_1 \cdot c_2$, changing the state of the second work qubit to $|c_1 \cdot c_2 \cdot c_3\rangle$. We continue applying Toffoli gates in this fashion, until the final work qubit is in the state $|c_1 \cdot c_2 \dots c_n\rangle$. Next, a U operation on the target qubit is performed, conditional on the final work qubit being set to one. That is, U is applied if and only if all of c_1 through c_n are set. Finally, the last part of the circuit just reverses the steps of the first stage, returning all the work qubits to their initial state, $|0\rangle$. The combined result, therefore, is to apply the unitary operator U to the target qubit, if and only if all the control bits c_1 through c_n are set, as desired. In the controlled gates we have been considering, conditional dynamics on the target qubit occurs if the control bits are set to one. Of course, there is nothing special about one, and it is often useful to consider dynamics which occurs conditional on the control bits being set to zero. For instance, suppose we wish to implement a two qubit gate in which the second ('target') qubit is flipped, conditional on the first ('control') qubit being set to zero. In figure 3.34 we introduce a circuit notation for this gate, together with an equivalent circuit in terms of the gates we have already introduced. Generically we shall use the open circle notation to indicate condition on the qubit being set to zero, while a closed circle indicates conditioning on the qubit being set to one. A more elaborate example of this connection, involving three control qubits, is illustrated in figure 3.35. The operation U is applied to the target qubit if the first and third qubits are set to zero, and the second qubit is set to one. It is easy to verify by inspection

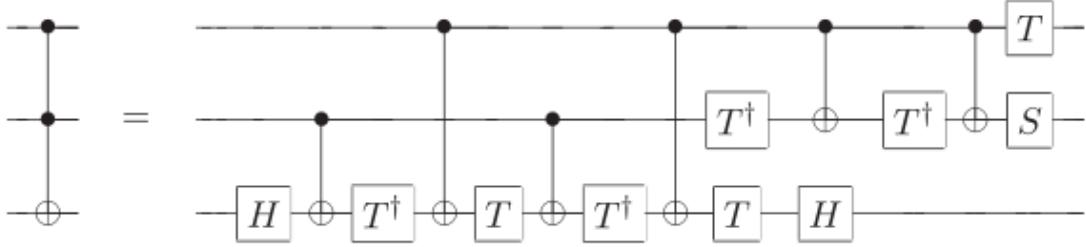


Figure 3.32: Implementation of the Toffoli gate using Hadamard, phase, controlled-NOT and $\pi/8$ gates

that the circuit on the right hand side of the figure implements the desired operation. More generally, it is easy to move between circuits which condition on qubits being set to one and circuit which condition on qubits being set to zero, by insertion of X gates in appropriate locations, as illustrated in figure 3.35. Another convention which sometimes is useful is to allow the controlled-NOT gates to have multiple targets as shown in figure 3.36. This natural notation means that when the control qubit is 1, then all the qubits marked with a \oplus are flipped, and otherwise nothing happens. It is convenient to use, for example, in construction classical functions such as permutations, or in encodes and decodes for quantum error-correction circuits, as we shall later see.

3.6 Universal quantum gates

A small set of gates (e.g. AND, OR, NOT) can be used to compute an arbitrary classical functions. We say that such a set of gates is universal for classical computation. In fact, since the Toffoli gate is universal for classical computation, quantum circuits subsume classical circuits. A similar universality is true for quantum computation, where a set of gates is said to be universal for quantum computation if any unitary operation may be approximated to arbitrary accuracy by a quantum circuit involving only those gates. We now describe three universality constructions for quantum computation. These constructions build upon each other, and culminate a proof that any unitary operation can be approximated to arbitrary accuracy using Hadamard, phase, CNOT, and $\pi/8$ gates. You may wonder why the phase gate appears in this list, since it can be constructed from two $\pi/8$ gates; it is included because of its natural role in fault-tolerant constructions described later.

The first construction shows that any arbitrary unitary operator may be ex-

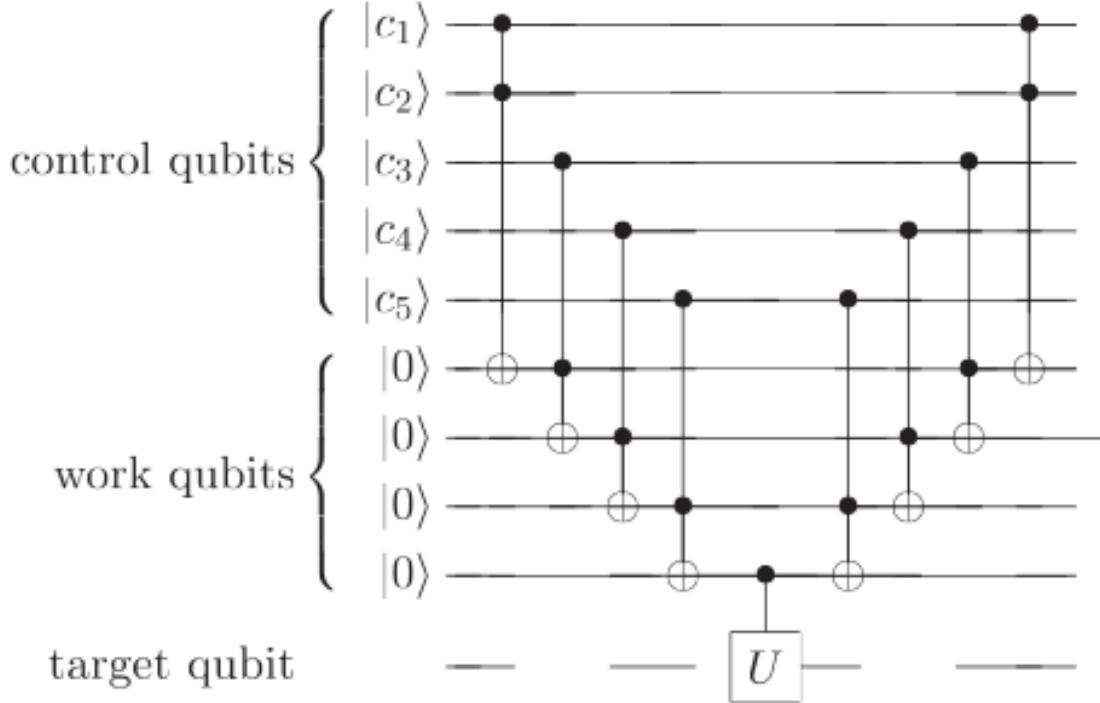


Figure 3.33: Network implementing the $C^n(U)$ operation for the case, $n = 5$

pressed exactly as a product of unitary operators that each acts non-trivially only on a subspace spanned by two computational basis states. The second construction combines the first construction with the results of the previous section to show that an arbitrary unitary operator may be expressed exactly using single qubit and CNOT gates. The third construction combines the second construction with a proof that single qubit operation may be approximated to arbitrary accuracy using the Hadamard, phase, and $\pi/8$ gates. This in turn implies that any unitary operation can be approximated to arbitrary accuracy using Hadamard, phase, CNOT and $\pi/8$ gates.

Our construction say little about efficiency - how many (polynomially or exponentially many) gates must be composed in order to create a given unitary transform. Later we show that there exist unitary transforms which require exponentially many gates to approximate. Of course, the goal of quantum computation is to find interesting families of unitary transformations that can be performed efficiently.

Example 3.6.1. Construct a quantum circuit to add two two-bit numbers x and y

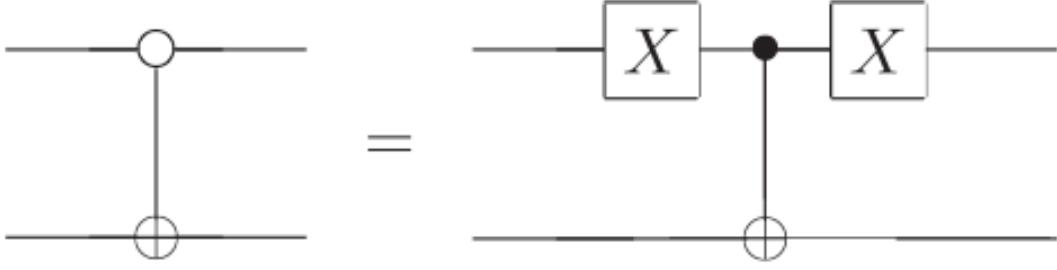


Figure 3.34: Controlled operation with a NOT gate being performed on the second qubit, conditional on the first qubit being set to zero

modulo 4. That is, the circuit should perform the transformation $|x, y\rangle \rightarrow |x, x + y \bmod 4\rangle$.

3.6.1 Two-level unitary gates are universal

Consider a Unitary matrix which acts on a d -dimensional Hilbert space. In this section we explain how U may be decomposed into a product of two-level unitary matrices; that is, unitary matrices which act non-trivially only on two-or-fewer vector components. The essential behind this decomposition may be understood by considering the case when U is 3×3 , so suppose U has the form

$$U = \begin{bmatrix} a & d & g \\ b & c & h \\ c & f & j \end{bmatrix}$$

We will find two-level unitary matrices U_1, \dots, U_3 such that

$$U_3 U_2 U_1 U = I$$

It follows that

$$U = U_1^\dagger U_2^\dagger U_3^\dagger$$

U_1, U_2 and U_3 are all two-level unitary matrices, and it is easy to see that their inverses, U_1^\dagger, U_2^\dagger and U_3^\dagger are also two-level unitary matrices. Thus, if we can demonstrate the above equations then we will have shown how to break U into a product of two-level unitary matrices. Use the following procedure to construct U_1 ; if $b = 0$ then set

$$U_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

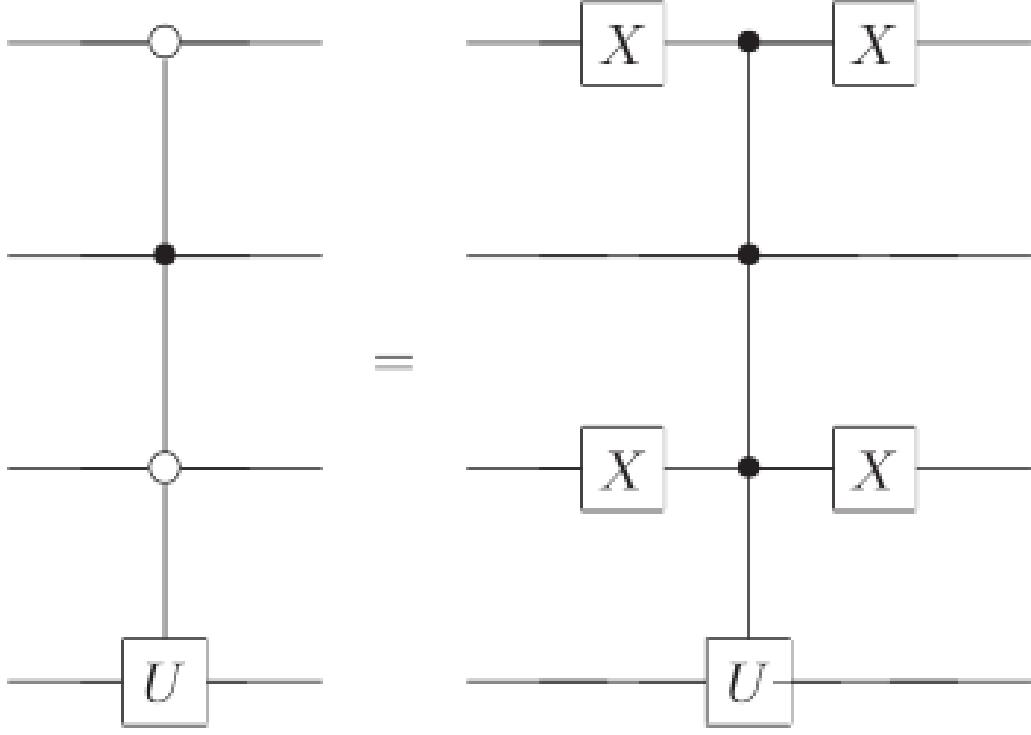


Figure 3.35: Controlled- U operation and its equivalent in terms of circuit elements we already know how to implement. The fourth qubit has U applied if the first and third qubits are set to zero, and the second qubit is set to one.

If $b \neq 0$ then set

$$U_1 = \begin{bmatrix} \frac{a^*}{\sqrt{|a|^2+|b|^2}} & \frac{b^*}{\sqrt{|a|^2+|b|^2}} & 0 \\ \frac{b}{\sqrt{|a|^2+|b|^2}} & \frac{-a}{\sqrt{|a|^2+|b|^2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Note that in either case U_1 is a two-level unitary matrix, and we multiply the matrices out we get

$$U_1 U = \begin{bmatrix} a' & d' & g' \\ 0 & e' & h' \\ c' & f' & j' \end{bmatrix}$$

The key point to note is that the middle entry in the left hand column is zero. We denote the other entries in the matrix with a generic prime ' ; their actually values



Figure 3.36: Controlled-NOT gate with multiple targets

do not matter.

Now apply a similar procedure to find a two-level matrix U_2 such that U_2U_1U has no entry in the bottom left corner. That is, if $c' = 0$ we set

$$U_2 = \begin{bmatrix} a'' & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

while if $c'' \neq 0$ then we set

$$\begin{bmatrix} \frac{a''}{\sqrt{|a'|^2+|c'|^2}} & 0 & \frac{c''}{\sqrt{|a'|^2+|c'|^2}} \\ 0 & 1 & 0 \\ \frac{c'}{\sqrt{|a'|^2+|c'|^2}} & 0 & \frac{-a'}{\sqrt{|a'|^2+|c'|^2}} \end{bmatrix}$$

In either case, when we carry out the matrix multiplication we find that

$$U_2U_1U = \begin{bmatrix} 1 & d''' & g''' \\ 0 & e''' & h''' \\ 0 & f''' & j''' \end{bmatrix}$$

Since U, U_1, U_2 are unitary, it follows that U_2U_1U is unitary, and thus $d''' = g''' = 0$, since the first row of U_2U_1U must have norm 1. Finally, set

$$U_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & e''' & f''' \\ 0 & h''' & j''' \end{bmatrix}$$

It is now easy to verify that $U_3U_2U_1U = I$, and thus $U = U_1^\dagger U_2^\dagger U_3^\dagger$, which is a decomposition of U into two-level unitaries.

More generally, suppose U acts on a d -dimensional space. Then, in a similar fashion to the 3×3 case, we can find two-level unitary matrices U_1, \dots, U_{d-1} such

that the matrix $U_{d-1}U_{d-2}\dots U_1U$ has a one in the top left hand corner, and all zeroes elsewhere in the first row and column. We then repeat this procedure for the $d-1$ by $d-1$ unitary submatrix in the lower righthand corner of $U_{d-1}U_{d-2}\dots U_1U$, and so on, with the end result that an arbitrary $d \times d$ unitary matrix may be written

$$U = V_1 \dots V_k$$

where the matrices V_i are two-level unitary matrices, and $k \leq (d-1) + (d-2) + \dots + 1 = d(d-1)/2$.

Example 3.6.2. Provide a decomposition of the transform

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \iota & -1 & -\iota \\ 1 & -1 & 1 & -\iota \\ 1 & -\iota & -1 & \iota \end{bmatrix}$$

into a product of two-level unitaries. This is a special case of the quantum Fourier transform.

A corollary of the above result is that an arbitrary unitary matrix on an n qubit system may be written as a product of at most $2^{n-1}(2^n - 1)$ two-level unitary matrices. For specific unitary matrices it may be possible to find much more efficient decompositions, but as you will now show there exist matrices which cannot be decomposed as a product of fewer than $d-1$ two-level unitary matrices!

Example 3.6.3. Prove that there exists a $d \times d$ unitary U which cannot be decomposed as a product of fewer than $d-1$ two level unitary matrices.

3.6.2 Single qubit and CNOT gates are universal

We have just shown that an arbitrary unitary matrix on a d -dimensional Hilbert space may be written as a product of two-level unitary matrices. Now we show that single qubit and CNOT gates together can be used to implement an arbitrary two-level unitary operation on the state space of n qubits. Combining these results we see that single qubit and CNOT gates can be used to implement an arbitrary unitary operation on n qubits, and therefore are universal for quantum computation.

Suppose U is a two-level unitary matrix on an n qubit quantum computer. Suppose in particular that U acts non-trivially on the space spanned by the computational basis states $|s\rangle$ and $|t\rangle$, where $s = s_1 \dots s_n$ and $t = t_1 \dots t_n$ are the binary

expansions for s and t . Let \tilde{U} be the non-trivial submatrix of U ; \tilde{U} can be thought of as a unitary operator on a single qubit.

Our immediate goal is to construct a circuit implementing U , built from single qubit and CNOT gates. To do this, we need to make use of Gray codes. Suppose we have distinct binary numbers, s and t . A Gray code connecting s and t is a sequence of binary numbers, starting with s and concluding with t , such that adjacent members of the list differ exactly in one bit. For instance, with $s = 101001$ and $t = 110011$ we have the Gray code

101001
101011
100011
110011

Let g_1 through g_m be the elements of a Gray code connecting s and t , with $g_1 = s$ and $g_m = t$. Note that we can always find a Gray code such that $m \leq n + 1$ since s and t can differ in at most n locations.

The basic idea of the quantum circuit implementing U is to perform a sequence of gates effecting the state changes $|g_1\rangle \rightarrow |g_2\rangle \rightarrow \dots \rightarrow |g_{m-1}\rangle$, then to perform a controlled- \tilde{U} operation, with the target qubit located at the single bit where g_{m-1} and g_m differ, and then to undo the first stage, transforming $|g_{m-1}\rangle \rightarrow |g_{m-2}\rangle \rightarrow \dots \rightarrow |g_1\rangle$. Each of these steps can be easily implemented using operations developed earlier in this chapter, and the final result is an implementation of U .

A more precise description of the implementation is as follows. The first step is to swap the states $|g_1\rangle$ and $|g_2\rangle$. Suppose g_1 and g_2 differ at the i th digit. Then we accomplish the swap by performing a controlled bit flip on the i th qubit, conditional on the values of the other qubits being identical to those in both g_1 and g_2 . Next we use a controlled operation to swap $|g_2\rangle$ and $|g_3\rangle$. We continue this fashion until we swap $|g_{m-2}\rangle$ with $|g_{m-1}\rangle$. The effect of this sequence of $m - 2$ operations is to achieve the operation

$$\begin{aligned}
&|g_1\rangle \rightarrow |g_{m-1}\rangle \\
&|g_2\rangle \rightarrow |g_1\rangle \\
&|g_3\rangle \rightarrow |g_2\rangle \\
&\vdots \\
&|g_{m-1}\rangle \rightarrow |g_{m-2}\rangle
\end{aligned}$$

All other computational basis states are left unchanged by this sequence of operations. Next, suppose g_{m-1} and g_m differ in the j th bit. We apply a controlled- \tilde{U} operation with the j th qubit as target, conditional on the other qubits having the same values as appear in both g_m and g_{m-1} . Finally, we complete the U operation by undoing the swap operations: we swap $|g_{m-1}\rangle$ with $|g_{m-2}\rangle$, then $|g_{m-2}\rangle$ with $|g_{m-3}\rangle$ and so on, until we swap $|g_2\rangle$ with $|g_1\rangle$.

A simple example illustrates the procedure further. Suppose we wish to implement the two-level unitary transformation

$$U = \begin{bmatrix} a & 0 & 0 & 0 & 0 & 0 & 0 & c \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ b & 0 & 0 & 0 & 0 & 0 & 0 & d \end{bmatrix}$$

Here, a, b, c and d are any complex numbers such that $\tilde{U} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$ is a unitary matrix.

Notice that U acts non-trivially only on the states $|000\rangle$ and $|111\rangle$. We write a gray code connection 000 and 111:

A	B	C
0	0	0
0	0	1
0	1	1
1	1	1

From this we read off the required circuit, shown in figure 3.37. The first two gates shuffle the states so that $|000\rangle$ gets swapped with $|011\rangle$. Next, the operation \tilde{U} is applied to the first qubit of the states $|011\rangle$ and $|111\rangle$, conditional on the second and third qubits being in the state $|11\rangle$. Finally, we unshuffle the states, ensuring that $|011\rangle$ gets swapped back with the state $|000\rangle$. Returning to the general case, we see that implementing the two-level unitary operation U requires at most $2(n - 1)$ controlled operations to swap $|g_1\rangle$ with $|g_{m-1}\rangle$ and then back again. Each of these controlled operations can be realized using $\mathcal{O}(n)$ single qubit and CNOT gates; the controlled- \tilde{U} operation also requires $\mathcal{O}(n)$ gates. Thus, implementing U requires

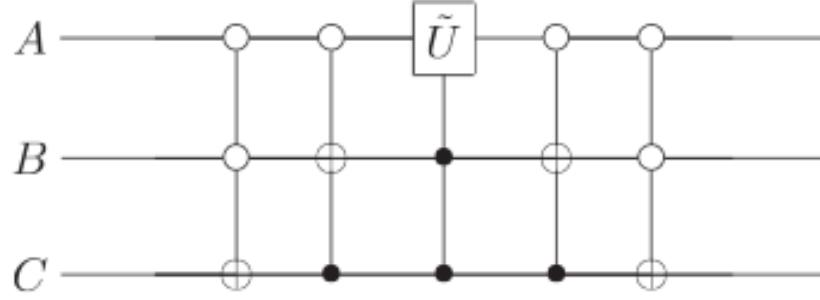


Figure 3.37: Circuit implementing the two-level unitary operation defined

$\mathcal{O}(n^2)$ single qubit and CNOT gates. We saw in the previous section that an arbitrary unitary matrix on the 2^n dimensional state space of n qubits may be written as a product of $\mathcal{O}(2^{2n}) = \mathcal{O}(4^n)$ two-level unitary operations. Combining these results, we see that an arbitrary unitary operation on n qubits can be implemented using a circuit containing $\mathcal{O}(n^2 4^n)$ single qubit and CNOT gates. Obviously, this construction does not provide terribly efficient quantum circuits!! However, we will later see that this construction is close to optimal in the sense that there are unitary operations that require an exponential number of gates to implement. Thus, to find fast quantum algorithm we will clearly need a different approach than is taken in the universality construction.

Example 3.6.4. Find a quantum circuit using single qubit operations and CNOTs to implement the transformation

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a & 0 & 0 & 0 & 0 & 0 & c \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & b & 0 & 0 & 0 & 0 & 0 & d \end{bmatrix}$$

where $\tilde{U} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$ is an arbitrary 2×2 unitary matrix.

3.6.3 Approximating arbitrary unitary gates

We've seen that any unitary transformation on n qubits can be built up out of a small set of elementary gates. Is it always possible to do this efficiently? That is, given a unitary transformation U on n qubits does there always exist a circuit of size polynomial in n approximating U ? The answer to this question turns out to be a resounding no: in fact, most unitary transformations can only be implemented very inefficiently. One way to consider the question: how many gates does it take to generate an arbitrary state on n qubits? A simple counting argument shows that this requires exponentially many operations, in general; it immediately follows that there are unitary operations requiring exponentially many operations. To see this, suppose we have g different types of gates available, and each work on at most f qubits. These numbers, f and g , are fixed by the computing hardware we have available, and may be considered to be constants. Suppose we have a quantum circuit containing m gates, starting from the computational basis state $|0\rangle^{\otimes n}$. For any particular gate in the circuit there are therefore at most $\binom{n}{f}^g = \mathcal{O}(n^{fg})$ possible choices. It follows that at most $\mathcal{O}(n^{fgm})$ different states may be computed using m gates. Suppose

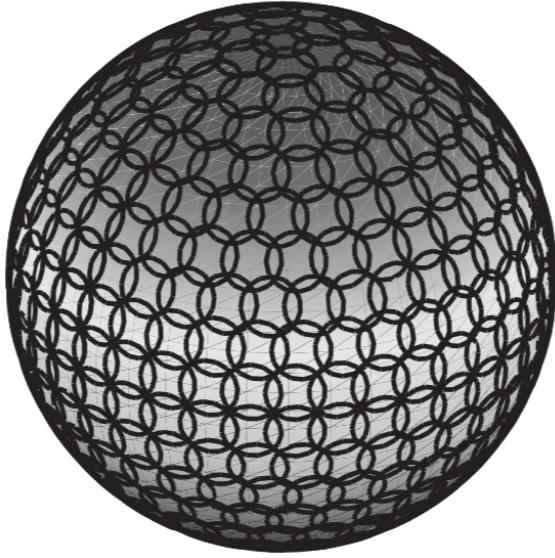


Figure 3.38: Visualization of covering the set of possible states with patches of constant radius

we wish to approximate a particular state, $|\psi\rangle$, to within a distance ϵ . The idea of the proof is to cover the set of all possible states with a collection of 'patches',

each of radius ϵ , and then to show that the number of patches required rises doubly exponentially in n ; comparing with the exponential number of different states that may be computed using m gates will imply the result. The first observation we need is that the space of state vectors of N qubits can be regarded as just the unit $2^{n+1} - 1$ sphere. To see this, suppose the n qubit states has amplitudes $\psi_j = X_j + iY_j$, where X_j and Y_j are the real and imaginary parts, respectively of the j th amplitude. The normalization condition for quantum states can be written $\sum_j (X_j^2 + Y_j^2) = 1$, which is just the condition for a point to be on the unit sphere in 2^{n+1} real dimensions, that is, the unit $(2^{n+1} - 1)$ sphere. Similarly, the surface area of radius ϵ near $|\psi\rangle$ is the same as the volume of a $(2^{n+1} - 2)$ sphere of radius ϵ . Using the formula $S_k(r) = 2\pi^{(k+1)/2}r^k/\Gamma((k+1)/2)$ for the surface area of a k -sphere of radius r , and $V_k(r) = 2\pi^{(k_1)/2}r^{k+1}/[(k+1)\Gamma((k+1)/2)]$ for the volume of a k -sphere of radius r , we see that the number of patches needed to cover the space goes like

$$S_{2^{n+1}-1}(a) = \frac{\sqrt{\pi}\Gamma(2^n - \frac{1}{2})(2^{n+1} - 1)}{\Gamma(2^n)\epsilon^{2^{n+1}-1}}$$

where Γ us the usual generalization of the factorial function. But $\Gamma(2^{n-1}/2) \geq \Gamma(2^n)/2^n$, so the number of patches required to cover the space is at least

$$\Omega\left(\frac{1}{\epsilon^{2^{n+1}-1}}\right)$$

Recall that the number of patches which can be reached in m gates if $\mathcal{O}(n^{fgm})$, so in order to reach all the ϵ -patches we must have

$$\mathcal{O}(n^{fgm}) \geq \Omega\left(\frac{1}{\epsilon^{2^{n+1}-1}}\right)$$

which gives

$$m = \Omega\left(\frac{2^n \log(1/\epsilon)}{\log(n)}\right)$$

That is, there are states of n qubits which take $\Omega(2^n \log(1/\epsilon)/\log(n))$ operations to approximate to within a distance ϵ . This is exponential in n , and thus is ‘difficult’ , in the sense of computational complexity . Furthermore, this immediately implies that there are unitary transformation on n qubits which take $\Omega(2^n \log(1/\epsilon)/\log(n))$ operations to approximate by a quantum circuit implementing an operation V such that $E(U, V) \leq \epsilon$. By contrast using our universality constructions and the Solovay-Kitaev theorem it follows that an arbitrary unitary operation U on n qubits may

be approximated to within a distance ϵ using $\mathcal{O}(n^2 4^n \log^c(n^2 4^n / \epsilon))$ gates. Thus, to within a polynomial factor the construction for universality we have given is optimal; unfortunately, it does not address the problem of determining which families of unitary operations can be computed efficiently in the quantum circuits model.

3.7 Measurements

A final element used in quantum circuits, almost implicitly sometimes, is measurement. In our circuits, we shall denote a projective measurement in the computational basis using a ‘meter’ symbol as illustrated in figure 3.39. In the theory of quantum circuits it is conventional to not use any special symbols to denote more general measurements, because, they can always be represented by unitary transformation with ancilla bits followed by projective measurements. There are two important principles

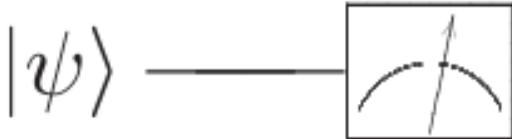


Figure 3.39: Symbol for projective measurement on a single qubit. In this circuit nothing further is done with the measurement results, but in more general quantum circuits it is possible to change later parts of the quantum circuit, conditional on measurement outcomes in earlier parts of the circuit. Such a usage of classical information is depicted using wires drawn with double lines (not shown here)

that is worth bearing in mind about quantum circuits. Both principles are rather obvious; however, they are of such great utility that they are worth emphasizing early. The first principle is that classically conditioned operations can be replaced by quantum conditioned operations:

3.7.1 Principle of deferred measurement

Definition 3.7.1. Measurement can always be moved from an intermediate stage of a quantum circuit to the end of the circuit; if the measurement results are used at any stage of the circuit then the classically controlled operations can be replaced by conditional quantum operations.

Often, quantum measurements are performed as an intermediate step in a quantum circuit, and the measurement results are used to conditionally control subsequent

quantum gates. That is the case, for example, in the teleportation circuit of figure 6.5. However, such measurements can always be moved to the end of the circuit. Figure 3.40 illustrates how this may be done by replacing all the classical conditional operations by corresponding quantum conditional operations, (Of course, some of the interpretation of this circuit as performing ‘teleportation’ is lost, because no classical information is transmitted from Alice to Bob, but it is clear that the overall action of the two quantum circuit is the same, which is the key point.) The second principle

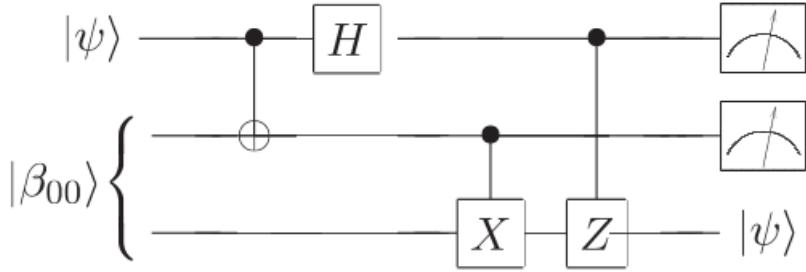


Figure 3.40: Quantum teleportation circuit in which measurements are done at the end, instead of in the middle of the circuit. The top two qubits belong to Alice, and the bottom one to Bob.

is even more obvious and surprisingly useful!

3.7.2 Principle of implicit measurement

Definition 3.7.2. Without loss of generality, any unterminated quantum wires (qubits which are not measured) at the end of a quantum circuit may be assumed to be measured.

To understand why this is true, imagine you have a quantum circuit containing just two qubits, and only the first qubit is measured at the end of the circuit. Then the measured statistics observed at this time are completely determined by the reduced density matrix of the first qubit. However, if a measurement had also been performed on the second qubit, then it would be highly surprising if that measurement could change the statistics of the measurement on the first qubit. This is because the reduced density matrix of the first qubit is not affected by performing a measurement on the second.

As you consider the role of measurements in quantum circuits, it is important to keep in mind that in its role as an interface between the quantum and classical

worlds, measurement is generally considered to be an irreversible operation, destroying quantum information and replacing it with classical information. In certain carefully designed cases however, this need not be true, as is vividly illustrated by teleportation and quantum error-correction. What teleportation and quantum-error correction have in common is that in neither instance does the measurement reveal any information about the identity of the quantum state being measured. Indeed, we will see in a later chapter that this is a more general feature of measurement - in order for a measurement to be reversible, it must reveal no information about the quantum state being measured!

Example 3.7.1. Suppose ρ is the density matrix describing a two qubit system. Suppose we perform a projective measurement in the computational basis of the second qubit. Let $P_0 |0\rangle\langle 0|$ and $P_1 = |1\rangle\langle 1|$ be the projectors onto the $|0\rangle$ and $|1\rangle$ states of the second qubit, respectively. Let ρ' be the density matrix which would be assigned to the system after the measurement by an observer who did not learn measurement result. Show that

$$\rho' = P_0 \rho P_0 + P_1 \rho P_1$$

Also show that the reduced density matrix for the first qubit is not affected by the measurement, that is, $\text{tr}(\rho) = \text{tr}(\rho')$.

Example 3.7.2. (Measurement in Bell basis) The measurement model we have specified for the quantum circuit model is that measurements are performed only in the computational basis. However, often we want to perform a measurement in some other basis, defined by a complete set of orthonormal states. To perform this measurement, simply unitarily transform from the basis we wish to perform the measurement in to the computational basis, then measure. For example, show that the circuit

performs a measurement in the basis of Bell states. More precisely, show that this circuit results in a measurement being performed with corresponding POVM elements the four projectors onto the Bell states. What are the corresponding measurement operators.

Example 3.7.3. (Measuring an operator) Suppose we have a single qubit operator U with eigenvalues ± 1 so that U is both Hermitian and unitary, os it can be regarded both as an observable and a quantum gate. Suppose we wish to measure the observable U . That is, we desire to obtain a measurement result indicating one of the two eigenvalues, and leaving a post measurement state which is the corresponding eigenvector. How can this be implementing by a quantum circuit? Show that the following circuit implements a measurement of U :

Example 3.7.4. (Measurement commuted with controls) A consequence of the principle of deferred measurement is that the measurements commute with quantum gates when the qubit being measured is a control qubit, that is:

(Recall that double lines represent classical bits in this diagram). Prove the first equality. The rightmost circuit is simply a convenient notation to depict the use of a measurement result to classically control a quantum gate.

Chapter 4

Quantum Circuits

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.

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 1 st 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).

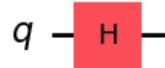
Note that if we have a circuit for unitary U , it is very easy to find a circuit for the inverse U^{-1} with the same complexity: just reverse the order of the gates, and take the inverse of each gate. For example, if $U = U_1 U_2 U_3$, then $U^{-1} = U_3^{-1} U_2^{-1} U_1^{-1}$.

A more efficient way to describe sequences of quantum operations and measurements, particularly for when the complexity of algorithms and protocols increases is through quantum circuit diagrams. The basic idea is as follows:

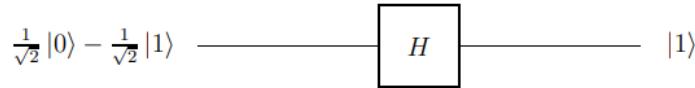
- Time goes from left to right
- Horizontal lines represent qubits
- Operations and measurements are represented by various symbols.

It is easiest to describe the model more specifically by using examples.

Example 4.0.1. The following diagram represents a Hadamard transform applied to a single qubit: If the input is $|\psi\rangle$, the output is $H|\psi\rangle$. Sometimes when we want



to explain what happens for a particular input, we label the inputs and outputs with superpositions, such as:



Example 4.0.2. Measurements are indicated by circuits (or ovals) with the letter M inside. For example: The result of measurement is a classical value, and sometimes (as in the above diagram) we draw double lines to indicate classical bits. In this case the outcome is fairly uniformly distributed bit.

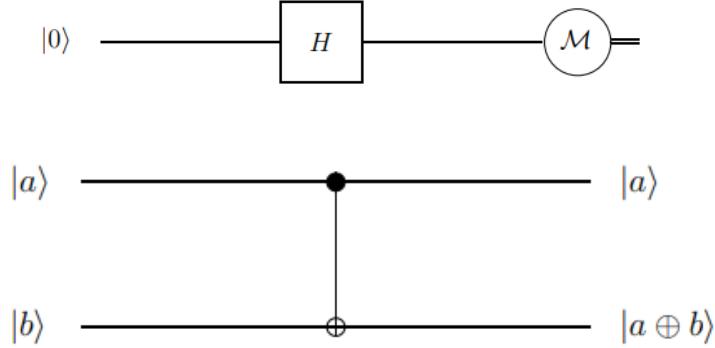


Figure 4.1: CNOT Gate

Example 4.0.3. Multiple-qubit gates are generally represented by rectangles, or have their own special representation. For instance controlled-not operation as shown in figure 4.1: Here the action is indicated for classical inputs (meaning $a, b \in \{0, 1\}$). Along similar lines, here is a controlled-controlled-not operations, better known as Toffoli gate as in figure 4.2. Here the action is described for each choice of $a, b, c \in \{0, 1\}$

Example 4.0.4. Usually we have multiple operations, such as in this circuit: It is not immediately obvious what this circuit does, so let us consider the superposition that would be obtained at various times for a selection of inputs: Suppose first that $|\psi_1\rangle = |00\rangle$. Then

$$\begin{aligned} |\psi_2\rangle &= \left(\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle \right) \left(\frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle \right) = \frac{1}{2} |00\rangle + \frac{1}{2} |01\rangle + \frac{1}{2} |10\rangle + \frac{1}{2} |11\rangle \\ |\psi_3\rangle &= \frac{1}{2} |00\rangle + \frac{1}{2} |01\rangle + \frac{1}{2} |11\rangle + \frac{1}{2} |10\rangle = \left(\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle \right) \left(\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle \right) \\ |\psi_4\rangle &= |00\rangle \end{aligned}$$

Next suppose that $|\psi_1\rangle = |01\rangle$. Then

$$\begin{aligned} |\psi_2\rangle &= \left(\left(\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle \right) \left(\frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle \right) \right) = \frac{1}{2} |00\rangle - \frac{1}{2} |01\rangle + \frac{1}{2} |10\rangle - \frac{1}{2} |11\rangle \\ |\psi_3\rangle &= \frac{1}{2} |00\rangle - \frac{1}{2} |01\rangle - \frac{1}{2} |10\rangle + \frac{1}{2} |11\rangle = \left(\frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle \right) \left(\frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle \right) \\ |\psi_4\rangle &= |11\rangle \end{aligned}$$

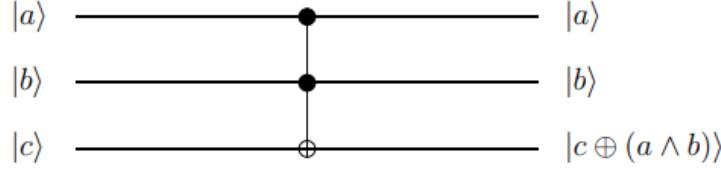
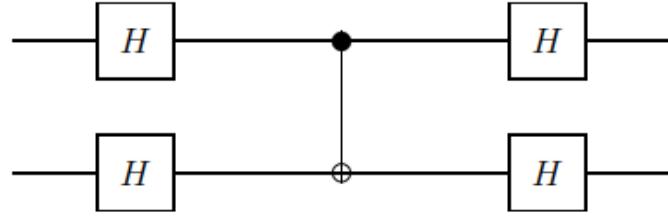


Figure 4.2: Toffoli Gate



Next suppose that $|\psi_1\rangle = |10\rangle$. Then

$$\begin{aligned} |\psi_2\rangle &= \left(\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \right) \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \right) \right) = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle - \frac{1}{2}|10\rangle - \frac{1}{2}|11\rangle \\ |\psi_3\rangle &= \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle - \frac{1}{2}|10\rangle - \frac{1}{2}|11\rangle = \left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \right) \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \right) \\ |\psi_4\rangle &= |10\rangle \end{aligned}$$

Next suppose that $|\psi_1\rangle = |11\rangle$. Then

$$\begin{aligned} |\psi_2\rangle &= \left(\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \right) \left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \right) \right) = \frac{1}{2}|00\rangle - \frac{1}{2}|01\rangle - \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \\ |\psi_3\rangle &= \frac{1}{2}|00\rangle - \frac{1}{2}|01\rangle - \frac{1}{2}|11\rangle + \frac{1}{2}|10\rangle = \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \right) \left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \right) \\ |\psi_4\rangle &= |01\rangle \end{aligned}$$

It turns out that the circuit is equivalent to this gate:

This may seem counter to your intuition because the roles of control and target effectively switched in the controlled-not gate because of the Hadamard transforms.

Example 4.0.5. Consider the quantum circuit as shown in figure 4.3.

1. Make Reverse circuit of the given circuit.

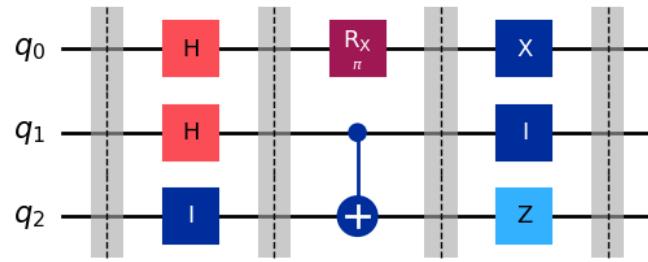
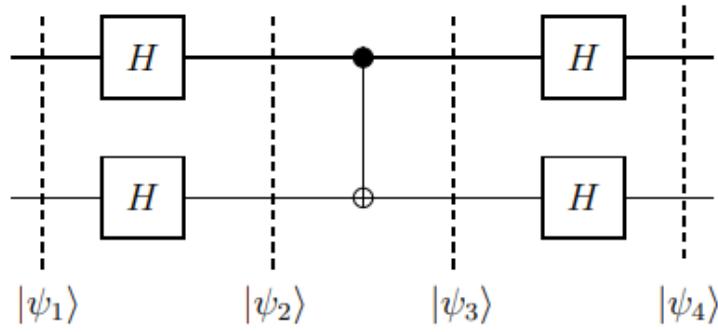


Figure 4.3: Quantum Circuit

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 4.4. 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 4.3 the circuit has been divided into parts for easy computation. At the first barrier, the input is $|010\rangle$. After applying the gates,

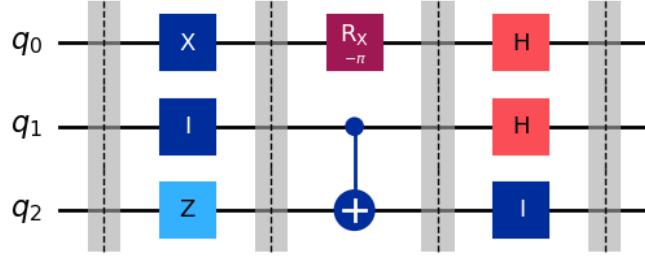


Figure 4.4: Reverse Quantum Circuit

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 complexte 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.

$$\begin{aligned} U &= S_3 S_2 S_1 \\ &= \frac{1}{2} \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 \\ -1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \end{bmatrix} \end{aligned}$$

4. 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.

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 of circuits and many more.

4.1 Classical Circuits simulating Quantum 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?

Classical circuits can simulate quantum circuits but the only known constructions have exponential overhead.

Theorem 4.1.1. *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 4.5. The initial state is a

α_{000}
α_{001}
α_{010}
\vdots
α_{111}

Figure 4.5: 2^n dimensional array storing the amplitudes of the state $\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$

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 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

4.2 Universal gate sets

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. The $\{NOT, AND\}$ is also a universal gate set.

In case of Quantum Computations as shown in previous chapter, any Unitary operator on n qubits can be implemented using 1- and 2-qubit gates.

Definition 4.2.1. A set of universal quantum gates S is universal if given any unitary operator U and desired precision ϵ 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 in quantum computing are $\{H, T, CNOT\}$, $\{H, Toffoli\}$.

Theorem 4.2.1. (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 ϵ 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 $\mathcal{O}(m \cdot \log^c(m/\epsilon))$. (Here polylog is the polynomial of the logarithm of the argument). All choices of Universal gates are equivalent*

Now we talk about universality of various elementary gate sets. 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 realistic 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 ϵ using a number of gates (from our small set) that is only *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.

4.2.1 A discrete set of universal operations

In the previous section we proved that the CNOT and single qubit unitaries together form a universal set for quantum computation. Unfortunately, no straightforward method is known to implement all these gates in a fashion which is resistant to errors. Fortunately, in this section, we find a discrete set of gates which can be used to perform universal quantum computation, and we'll show how to perform these gates in an error resistant fashion, using quantum error-correcting codes.

Approximating Unitary operation

Obviously, a discrete set of gates can't be used to implement an arbitrary unitary operation exactly, since the set of unitary operations is continuous. Rather, it turns out that a discrete set can be used to approximate any unitary operation. To understand how this works, we first need to study what it means to approximate a unitary operation. Suppose U and V are two unitary operations on the same state space. U is the target unitary operation that we wish to implement, and V is the unitary operator that is actually implemented in practice. We define the error when V is implemented instead of U by

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

where the maximum is over all normalized quantum states $|\psi\rangle$ in the state space. We will now show that this measure of error has the interpretation that if $E(U, V)$ is

small, then any measurement performed on the state $V|\psi\rangle$ will give approximately the same measurement statistics as a measurement of $U|\psi\rangle$, for any initial state $|\psi\rangle$. More precisely, we show that if M is a POVM element in an arbitrary POVM, and P_U (or P_V) is the probability of obtaining this outcome if U (or V) were performed with a starting state $|\psi\rangle$ then

$$|P_U - P_V| \leq 2E(U, V)$$

Thus, if $E(U, V)$ is small, then measurement outcomes occur with similar probabilities, regardless of whether U or V were performed. Also if we perform a sequence of gates V_1, \dots, V_m intended to approximate some other sequence of gates U_1, \dots, U_m , then the errors add at most linearly,

$$E(U_m U_{m-1} \dots U_1, V_m V_{m-1} \dots V_1) \leq \sum_j -j = 1^m E(U_j, V_j)$$

The approximation results are very useful. Suppose we wish to perform a circuit containing m gates U_1 through U_m . Unfortunately, we are only able to approximate the gate U_j by the gate V_j . In order that the probabilities of different measurement outcomes obtained from the approximate circuit be within a tolerance $\Delta > 0$ of the correct probabilities, it suffices that $E(U_j, V_j) \leq \Delta/(2m)$, by the results.

Universality of Hadamard, Phase, CNOT and $\pi/8$ gates

We're now in a good position to study the approximation of arbitrary unitary operations by discrete set of gates. We're going to consider two different discrete set of gates, both of which are universal. The first set, the standard set of universal gates, consist of Hadamard, phase, Controlled-NOT and $\pi/8$ gates. We provide fault-tolerant constructions for these gates; they also provide an exceptionally simple universality construction. The second set of gates we consider consists of the Hadamard gate, phase gate, the controlled-NOT gate, and the Toffoli gate. These gates can also all be done fault-tolerantly; however, the universality proof and fault-tolerance construction for these gates is a little less appealing.

We being the universality proof by showing that the Hadamard and $\pi/8$ gates can be used to approximate any single qubit unitary operation to arbitrary accuracy. Consider the gates T and HTH . T is up to an unimportant global phase, a rotation by $\pi/4$ radians around the \hat{z} axis on the Bloch sphere, while HTH is a rotation by $\pi/4$ radians around the \hat{x} axis on the Bloch sphere. Composing these two operations

gives, up to a global phase,

$$\begin{aligned} e^{-iota \frac{\pi}{8} Z} e^{-\iota \frac{\pi}{8} X} &= \left(\cos \frac{\pi}{8} I - \iota \sin \frac{\pi}{8} X \right) \left(\cos \frac{\pi}{8} I - \iota \sin \frac{\pi}{8} X \right) \\ &= \cos^2 \frac{\pi}{8} I - \iota \left(\cos \frac{\pi}{8} (X + Z) + \sin \frac{\pi}{8} Y \right) \sin \frac{\pi}{8} \end{aligned}$$

This is a rotation of the Bloch sphere about an axis along $\vec{n} = (\cos \frac{\pi}{8}, \sin \frac{\pi}{8}, \cos \frac{\pi}{8})$ with corresponding unit vector \hat{n} , and through an angle θ defined by $\cos(\theta/2) = \cos^2 \frac{\pi}{8}$. That is, using only the Hadamard and $\pi/8$ gates we can construct $R_{\hat{n}}(\theta)$. Moreover, this θ can be shown to be an irrational multiple of 2π .

Next, we show that repeated iteration of $R_{\hat{n}}(\theta)$ can be used to approximate to arbitrary accuracy any rotation $R_{\hat{n}}(\alpha)$. To see this, let $\delta > 0$ be the desired accuracy, and let n be an integer larger than $2\pi/\delta$. Define θ_k so that $\theta \in [0, 2\pi)$ and $\theta_k = (k\theta) \text{mod} 2\pi$.

4.2.2 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 ϵ then the following *Hybrid Argument*.

Proposition 4.2.2. *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:

$$\begin{aligned} U_K \dots U_2 U_1 - \tilde{U}_K \dots \tilde{U}_1 &= U_K \dots U_2 U_1 - U_K \dots U_2 \tilde{U}_1 + U_K \dots U_2 \tilde{U}_1 \\ &\quad - U_K \dots \tilde{U}_2 \tilde{U}_1 + \dots + U_K \tilde{U}_{K-1} \dots \tilde{U}_1 - \tilde{U}_K \dots \tilde{U}_1 \\ &= U_K \dots U_2 (U_1 - \tilde{U}_1) + U_K \dots U_3 (U_2 - \tilde{U}_2) \tilde{U}_1 + \dots \\ &\quad + U_K (U_{K-1} - \tilde{U}_{K-1}) \dots \tilde{U}_1 + (U_K - \tilde{U}_K) \dots \tilde{U}_1 \end{aligned}$$

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 ϵ , 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

Important Note

Suppose a quantum system starts in the state $|\psi\rangle$, and we perform either the unitary operation U , or the unitary operation V . Following this, we perform a measurement. let M be a POVM element associated with the measurement, and let P_U (or P_V) be the probability of obtaining the corresponding measurement outcome if the operation U (or V) was performed. Then

$$|P_U - P_V| = |\langle\psi| U^\dagger M U |\psi\rangle - \langle\psi| V^\dagger M V |\psi\rangle|$$

Let $|\Delta\rangle = (U - V)|\psi\rangle$, Simple algebra and the Cauchy-Schwarz inequality show that

$$\begin{aligned} |P_U - P_V| &= |\langle\psi| U^\dagger M |\Delta\rangle + \langle\Delta| M V |\psi\rangle| \\ &\leq |\langle\psi| U^\dagger M |\Delta\rangle| + |\langle\Delta| M V |\psi\rangle| \\ &\leq \|\Delta\| + \|\Delta\| \\ &\leq 2E(U, V) \end{aligned}$$

The inequality $|P_U - P_V| \leq 2E(U, V)$ gives quantitative expression to the idea that when the error $E(U, V)$ is small, the difference between measurement outcomes is also small.

Suppose we perform a sequence V_1, V_2, \dots, V_m of gates intended to approximate some other sequence of gates, U_1, U_2, \dots, U_m . Then it turns out that the error caused by the entire sequence of imperfect gates is at most the sum of the errors in the individual gates,

$$E(U_m U_{m-1} \dots U_1, V_m V_{m-1} \dots V_1) \leq \sum_{j=1}^m E(U_j, V_j)$$

To prove this we start with the case $m = 1$. Note that for some $|\psi\rangle$ we have

$$\begin{aligned} E(U_2 U_1, V_2 V_1) &= \|(U_2 U_1 - V_2 V_1)|\psi\rangle\| \\ &= \|(U_2 U_1 - V_2 U_1)|\psi\rangle + (V_2 U_1 - V_2 V_1)|\psi\rangle\| \end{aligned}$$

Using the triangle inequality $\|a\rangle + |b\rangle\| \leq \|a\rangle\| + \|b\rangle\|$, we obtain

$$\begin{aligned} E(U_2 U_1, V_2 V_1) &\leq \|(U_2 - V_2)U_1|\psi\rangle\| + \|V_2(U_1 - V_1)|\psi\rangle\| \\ &\leq E(U_2, V_2) + E(U_1, V_1) \end{aligned}$$

which was the desired result. The result for general m follows by induction.

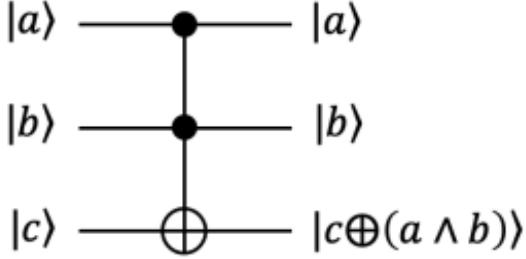


Figure 4.6: Toffoli gate acting on computational basis states ($a, b, c \in \{0, 1\}$)

4.3 Reversible Computation

Any classical circuit can be asymptotically efficiently implemented using a quantum circuit. (Classical gates can be efficiently simulated using quantum circuits.)

4.3.1 Quantum Circuits simulating Classical Circuits

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 4.6. 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 8×8 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 8×8

matrix. The Toffoli gate is sometimes called the controlled-controlled-NOT gate or controlled-CNOT gate.

Theorem 4.3.1. *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 4.7. Bits are represented as computational basis

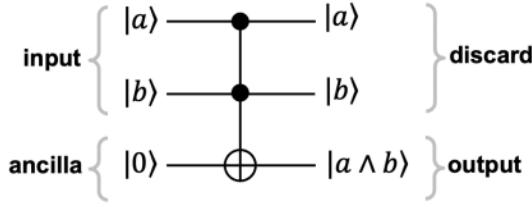


Figure 4.7: Using a Toffoli gate to simulate classical AND gate

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 4.7. This causes the state of the ancilla qubit to become $|a \wedge b\rangle$. 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 4.8. 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 2×2 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 “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

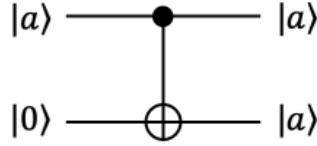


Figure 4.8: Using a CNOT gate to simulate a classical fan-out gate

as V. Henceforth, assumed 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^\dagger gate. Now, consider the following circuit in figure 4.9, consisting of 2-qubit gates. This computes Toffoli

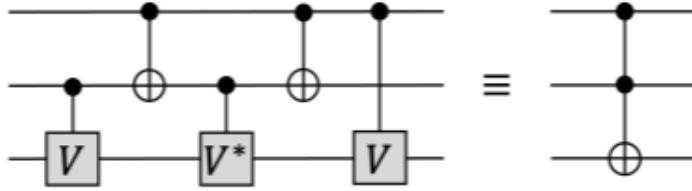


Figure 4.9: Simulating a Toffoli gate in terms of 2-qubit gates

gate. we can work out the 8×8 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 $|00\rangle, |01\rangle, |10\rangle, |11\rangle$.

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 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

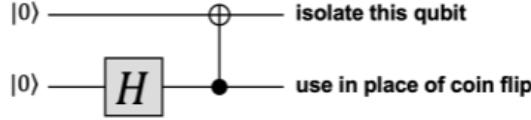


Figure 4.10: Simulating random bits without using measurements

circuits is measured at the end will be exactly the same as if a random bit had been inserted at that place in the computation. \square

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 4.3.2. *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.*

4.3.2 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).

$$a = \frac{k}{2^m} = \sum_{i \in [n]} k_i 2^{i-m} = (k_{n-1} \dots k_m.k_m k_{m-1} \dots k_0)$$

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 $(0.k_{n-1} \dots k_i)$ is called n -bit fixed point representation of a . Therefore to represent a the additive precision ϵ , $n = \lceil \log_2(\epsilon) \rceil$ 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 $\mathcal{O}(\text{poly}(n))$. 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.

4.4 Energy and computation

Computational complexity theory studies the amount of time and space required to solve a computational problem. Another important computational resource is energy. In this section, we study the energy requirements for computational. Surprisingly, it turns out that computation, both classical and quantum, can in principle be done without expending any energy! Energy consumption in computation turns out to be deeply linked to the reversibility of computation. Consider a gate like the NAND gate, which takes an input two bits, and produces a single bit as output. This gate is intrinsically irreversible because, given the output of the gate, the input is not uniquely determined. For example, if the output of the NAND gate is 1, then the input could have been any one of 00, 01 or 10. On the other hand, the NOT gate is an example of a reversible logic gate because, given the output of the NOT gate, it is possible to infer what the input must have been.

Another way of understanding irreversibility is to think of it in terms of information erasure. If a logic gate is irreversible, then some of the information input to the gate is lost irretrievably when the gate opens -that is, some of the information has been erased by the gate. Conversely, in a reversible computation, no information is ever erased, because the input can always be recovered from the output. Thus, saying that a computation is reversible is equivalent to saying that no information is erased during the computation.

What is the connection between energy consumption and irreversibility in computation? Landauer's principle provides the connection, stating that, in order to erase information, it is necessary to dissipate energy. More precisely, Landauer's principle may be stated as follows:

Definition 4.4.1. Landauer's principle (first form): Suppose a computer erases a single bit of information. The amount of energy dissipated into the environment is at least $k_B T \ln 2$, where k_B is a universal constant known as Boltzmann's constant, and T is the temperature of the environment of the computer.

According to the laws of thermodynamics, Landauer's principle can be given an alternative form stated not in terms of energy dissipation, but rather in terms of entropy.

Definition 4.4.2. Landauer's principle (second form): Suppose a computer erases a single bit of information. The entropy of the environment increases by at least $k_b T \ln 2$, where k_b is Boltzmann's constant.

Justifying Landauer's principle is a problem of physics. If we accept Landauer's principle as given, then it raises a number of interesting questions. First of all, Landauer's principle only provides a lower bound on the amount of energy that must be dissipated to erase information. How close are existing computers to this lower bound? Not very, turns out to be the answer - computers circa the year 2000 dissipate roughly $500k_n T \ln 2$ in energy for each elementary logical operation.

Although existing computers are far from the limit set by Landauer's principle, it is still an interesting problem of principle to understand how much the energy consumption can be reduced. Aside from the intrinsic interest of the problem, a practical reason for the interest follows from Moore's law: if computer power keeps increasing them the amount of energy dissipated must also increases unless the energy dissipated per operation drops at least as fast as the rate of increasing computing power.

If all computations could be done reversibly, then Landauer's principle would imply no lower bound on the amount of energy dissipated by the computer, since no bits at all are erased during a reversible computation. Of course, it is possible that some other physical principle might require that energy dissipated during the computation: fortunately, this turns out not to be the case. But is it possible to perform universal computation without erasing any information? Physicists can cheat on this problem to see in advance that the answer to this question must be yes, because our present understanding of the laws of physics is that they are fundamentally reversible. That is, if we know the final state of a closed physical system, then the laws of physics allows us to work out the initial state of the system. If we believe that those laws are correct, then we must conclude that hidden in the irreversible logic gates like AND and OR, there must be some underlying reversible computation. But where is the hidden reversibility, and can we use it to construct manifestly reversible computers?

We will use two different techniques to give reversible circuit-based models capable of universal computation. The first model, a computer built entirely of billiard balls and mirrors, gives a beautiful concrete realization of the principles of reversible computation. The second model, based on a reversible logic gate known as the Toffoli gate, is a more abstract view of reversible computation. It is also possible

to build reversible Turing machines that are universal for computation; however we won't study these here, since the reversible circuit models turns out to be much more useful for quantum computation.

4.4.1 The billiard ball model of computation

The basic idea of the billiard ball computer is illustrated in figure 4.11. Billiard ball 'inputs' enter the computer from the left hand side, bouncing off mirrors and each other, before exiting as 'outputs' on the right hand side. The presence or absence of a billiard ball at a possible input side is used to indicate a logical 1 or a logical 0, respectively. The fascination thing about this model is that it is manifestly reversible, in so far its operation is based on the laws of classical mechanics. Furthermore, this model of computation turns out to be universal in the sense that it can be used to simulate an arbitrary computation in the standard circuit model of computation. Of course, if a billiard ball computer were ever built it would be highly

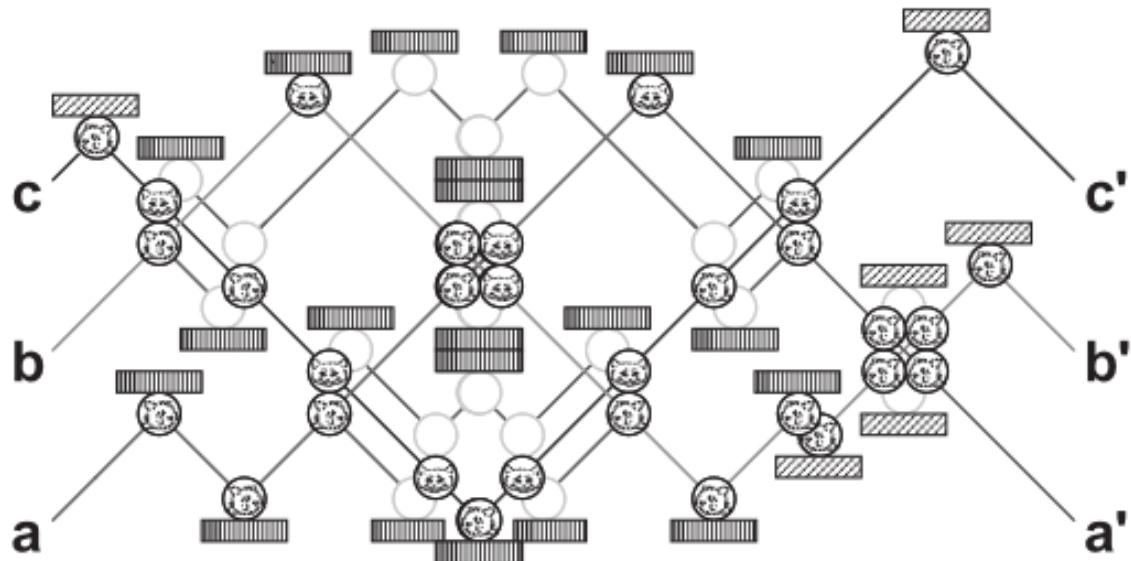


Figure 4.11: A simple billiard ball computer, with three input bits and three output bits, shown entering on the left and leaving on the right, respectively. The presence or absence of a billiard ball indicates a 1 or a 0, respectively. Empty circles illustrate potential paths due to collisions. This particular computer implements the Fredkin classical reversible logic gate, discussed in the text.

unstable. As any billiard player can attest, a billiard ball rolling frictionlessly over a smooth surface is easily knocked off course by small perturbations. The billiard ball model of computation depends on perfect operation, and the absence of external perturbations such as those caused by thermal noise. Periodic corrections can be performed, but information gained by doing this would have to be erased, requiring work to be performed. Expenditure of energy thus serves the purpose of reducing this susceptibility to noise, which is necessary for practical, real-world computational machine. For this purpose of this introduction we will ignore the effects of noise on the billiard ball computer, and concentrate on understanding the essential elements of reversible computation.

The billiard ball computer provides an elegant means for implementing a reversible universal logic gate known as the Fredkin gate. Indeed, the properties of the Fredkin gate provide an informative overview of the general principles of reversible logic gates and circuits. The Fredkin gate has three input bits and three output bits, which we refer to as a, b, c and a', b', c' , respectively. The bit c is a control bit, whose value is not changed by the action of the Fredkin gate, that is, $c' = c$. The reason c is called the control bit is because it controls what happens to the other two bits, a and b . If c is set to 0 then a and b are left alone, $a' = a$ and $b' = b$. If c is set to 1, a and b are swapped, $a' = b$, $b' = a$. The explicit truth table for the Fredkin gate is shown in table 4.12. It is easy to see that the Fredkin gate is reversible, because given the output a', b', c' , we can determine the inputs a, b, c . In fact, to recover the original inputs a, b and c we need only apply another Fredkin gate to a', b', c' :

Example 4.4.1. (Fredkin gate is self-inverse) Show that by applying two consecutive Fredkin gates gives the same outputs as inputs.

Examining the parts of the billiard balls in figure 4.11, it is not difficult to verify that this billiard ball computer implements the Fredkin gate.

Example 4.4.2. Verify that the billiard ball computer in figure 4.11 computes the Fredkin gate.

In addition to reversibility, the Fredkin gate also has the interesting property that the number of 1s is conserved between the input and output. In terms of the billiard ball computer, this corresponds to the number of billiard balls going into the Fredkin gate being equal to the number coming out. Thus, it is sometimes referred to as being a conservative reversible logic gate. Such reversibility and conservative properties are interesting to a physicist because they can be motivated by fundamental physical principles. The laws of Nature are reversible, with the possible exception of the measurement postulate of quantum mechanics. The conservative property can be

Inputs			Outputs		
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	1	0	1
1	0	0	1	0	0
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	1	1	1

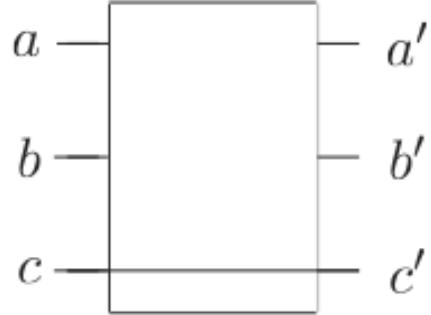


Figure 4.12: Fredkin gate truth table and circuit representation. The bits a and b are swapped if the control bit c is set, and otherwise are left alone

through of as analogous to properties such as conservation of mass, or conservation of energy. indeed, the billiard ball model of computation the conservative property corresponds exactly to conservation of mass.

The Fredkin gate is not only reversible and conservative, it's a universal logic gate as well!!.. 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 “ancilla bit” and then applying the reversible gate. One of the ways is to store all the intermediate steps of the computation. On the quantum computer, storing all the intermediate steps is problematic because of waste of quantum resources as those qubits can then be never used again, and intermediate bits stored in some extra qubits are still entangled to the quantum state of interest. (If the environment interferes with the intermediate result, then the quantum state of interest also gets affected).

As illustrated in figure 4.13, the Fredkin gate can be configured to simulate AND, NOT, CROSSOVER and FANOUT functions, and thus can be cascaded to simulate any classical circuit whatsoever. To simulate irreversible gates such as AND using the Fredkin gate, we made use of two ideas. First, we allowed the input of ‘ancilla’ bits to the Fredkin gate, in specially prepared states, either 0 or 1. Second, the output

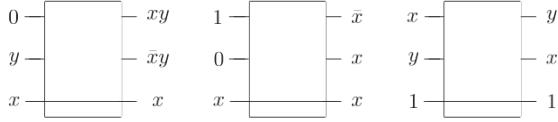


Figure 4.13: Fredkin gate configured to perform the elementary gates AND (left), NOT(middle), and a primitive routing function, the CROSSOVER (right). The middle gate also serves to perform the FANOUT operation, since it produces two copies of x at the output. Note that each of these configuration requires the use of extra ‘ancilla’ bits prepared in standard states - for example, the input 0 input on the first line of the AND gate - and in general the output contains ‘garbage’ not needed for the remainder of the computation.

of the Fredkin gate contained extraneous ‘garbage’ not needed for the remainder of the computation. The ancilla and garbage bits are not directly important to the computation. Their importance lies in the fact that they make the computation reversible. Indeed the irreversibility of gates like the AND and OR may be viewed as a consequence of the ancilla and garbage bits being ‘hidden’. Summarizing, given any classical circuit computing a function $f(x)$, we can build a reversible circuit made entirely of Fredkin gates, which on input of x , together with some ancilla bits in a standard state a , computes $f(x)$, together with some extra ‘garbage’ output, $g(x)$. Therefore, we represent the action of the computation as $(x, a) \rightarrow (f(x), g(x))$.

We now know how to compute function reversibly. Unfortunately, this computation produces unwanted garbage bits $g(x)$. With some modifications it turns out to be possible to perform the computation so that any garbage bits produced are in a standard state. This construction is crucial for quantum computation, because garbage bits whose values depends upon x will in general destroy the interference properties crucial to quantum computation. To understand how this works it is convenient to assume that the NOT gate is available on our repertoire of reversible gates, so we may as well assume that the ancilla bits a all start out as 0s, with NOT gate being added where necessary to turn the ancilla 0s into 1s. It will also be convenient to assume that the classical controlled-NOT gate is available, that is, the inputs (c, t) are taken to $(c, t \oplus c)$, where \oplus denotes addition modulo 2. Notice that $t = 0$ gives $(c, 0) \rightarrow (c, c)$, so the controlled-NOT can be thought of as a reversible copying gate or FANOUT, which leaves no garbage bits at the output.

With the additional NOT gates appended at the beginning of the circuit, the action of the computation may be written as $(x, 0) \rightarrow (f(x), g(x))$. We could also add CNOT gates to the beginning of the circuit, in order to create a copy of x which

is not changed during the subsequent computation. With this modification, the circuit may be written as

$$(x, 0, 0) \rightarrow (x, f(x), g(x))$$

This is a very useful way of writing the action of the reversible circuit, because it allows an idea known as uncomputation to be used to get rid of the garbage bits, for a small cost in the running time of the computation. The idea is the following. Suppose we start with a four register computer in the state $(x, 0, 0, y)$. The second register (also called ancilla qubits) is used to store the result of the computation, and the third register is used to provide workspace for the computation (also called working register aka garbage register - they are ancilla registers which can be freed after uncomputation), that is, the garbage bits $g(x)$. The use of the fourth register is described shortly, and we assume it starts in an arbitrary state y .

We begin as before, by applying a reversible circuit to compute f , resulting in the state $(x, f(x), g(x), y)$. From no deletion theorem, there is no generic unitary operator that can set a black-box state to 0. We wish to set the working registers back to 0 while keeping the input and output state. Next, we use CNOTs to add the result $f(x)$ bitwise to the fourth register, leaving the machine in the state $(x, f(x), g(x), y \oplus f(x))$. Note that here the CNOT gates only perform the classical XOR operation (or classical copying operation if $y = 0$ in the computational basis) and thus does not violate the *No-Cloning Theorem*. Note that all the steps used to compute $f(x)$ were reversible ($U_f^\dagger = U_f^{-1}$) and did not affect the fourth register, so by applying the reverse of the circuit used to compute f we come to the state $(x, 0, 0, y \oplus f(x))$. We can then apply a SWAP operation on the working registers and the ancilla registers to obtain $(x, 0, 0, y \oplus f(x)) \rightarrow (x, y \oplus f(x), 0, 0)$. Thus, note that the working registers (aka garbage registers) are set to 0 and are no longer entangled to the input or output register and can be reused for other purposes. This is called **Uncomputation**. Thus, after the uncomputation step the second and third registers as shown are unchanged before and after the application of the circuit, though they are changed during the intermediate steps. Typically, we omit the ancilla 0s from the description of the function evaluation, and just write the action of the circuit as

$$(x, y) \rightarrow (x, y \oplus f(x))$$

allowing us to simplify notation and focus on the essence of Quantum Algorithm. In general we refer to this modified circuit computing f as the reversible circuit computing f , even though in principle there are many other reversible circuits which could be used to compute f . The reversible gate is thus defined as $(x, y) \rightarrow (x, f(x)) \oplus$

y). Thus, in particular we have $(x, 0) \rightarrow (x, f(x))$ where 0 are the ancilla qubits required. Note that we have not considered the working register here because they can be freed after the computation via uncomputation and thus can be further used in the computation.

What resource overhead is involved in doing reversible computation? To analyze this question, we need to count the number of extra ancilla bits needed in a reversible circuit, and compare the gate counts with classical models. It ought to be clear that the number of gates in a reversible circuit is same as in an irreversible circuit to within the constant factor which represents the number of Fredkin gates needed to simulate a single element of the irreversible circuit, and an additional factor of two for uncomputation with an overhead for the extra CNOT operation used in reversible computation which is linear in the number of bits involved in the circuit. Similarly, the number of ancilla bits required scales at most linearly with the number of gates in the irreversible circuit, since each element in the irreversible circuit can be simulated using a constant number of ancilla bits. As a result, natural complexity classes such as P and NP are the same not matter whether a reversible or irreversible model of computation is used.

Thus, 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.

$$U_f : \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x, 0\rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x, f(x)\rangle$$

This does not necessarily 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.

Example 4.4.3. (Reversible half-adder) Construct a reversible circuit, when two bits x and y are input, outputs $(x, y, c, x \oplus y)$, where c is the carry bit when x and y are added.

The Fredkin gate and its implementation using the billiard ball computer offers a beautiful paradigm for reversible computation. There is another reversible logic gate, the Toffoli gate, which is also universal for classical computation. While the Toffoli gate does not have qubit the same elegant physical simplicity as the billiard ball implementation of the Fredkin gate, it will be more useful in the study of quantum computation. For convenience, we review its properties here. The Toffoli gate has

Inputs			Outputs		
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

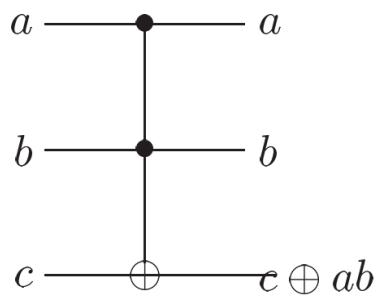


Figure 4.14: Truth table and circuit representation for the Toffoli Gate

three input bits, a , b and c . a and b are known as the first and second control bits, where c is the target bit. The gate leaves both control bits unchanged, flips the target bit if both control bits are set, and otherwise leaves the target bit alone. The truth table and the circuit representation for the Toffoli gate are as shown in the figure 4.14. How can the Toffoli gate be used to do universal computation? Suppose we wish to NAND the bits a and b . To do this using the Toffoli gate, we input a and b as control bits, and send in an ancilla bit set to 1 as the target bit, as shown in figure 4.15. The NAND of a and b is output as the target bit. As expected from our study of the Fredkin gate, the Toffoli gate simulation of a NAND requires the use of a special ancilla input, and some of the outputs from the simulation are garbage bits. The Toffoli gate can also be used to implement the FANOUT operation by inputting an ancilla 1 to the first control bit, and a to the second control bit, producing the output 1, a , a . This is illustrated in figure 4.16. Recalling that NAND and FANOUT are together universal for computation, we see that an arbitrary circuit can be efficiently simulated using a reversible circuit consisting only of Toffoli gates and ancilla bits, and that useful additional techniques such as uncomputation may be achieved using the same methods as were employed with the Fredkin gate.

Our interest in reversible computation was motivated by our desire to understand

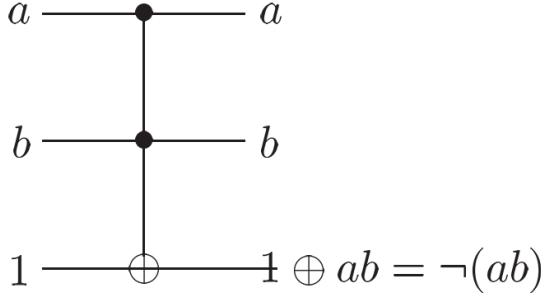


Figure 4.15: Implementing NAND gate using a Toffoli gate. The top two qubits represent the input to the NAND, while the third bit is prepared in the state 1, sometimes known as the ancilla state. The output from the NAND is in the third bit.

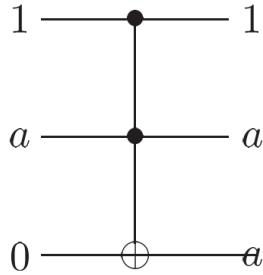


Figure 4.16: FANOUT with the Toffoli gate, with the second bit being the input to the FANOUT, and the other two bits standard ancilla bits. The output from the FANOUT appears on the second and the third bits.

the energy requirements for computation. It is clear that the noise-free billiard ball model of computation requires no energy for its operation; what about models based upon the Toffoli gate? This can only be determined by examining specific models for the computation of the Toffoli gate. It turns out that, indeed, the Toffoli gate can be implemented in a manner which does not require expenditure of energy.

There is a significant caveat attached to the idea that computation can be done without the expenditure of energy. As we noted earlier, the billiard ball model of computation is highly sensitive to noise, and this is true of many other models of reversible computation. To nullify the effects of noise, some form of error-correction needs to be done. Such error-correction typically involves the performance of measurements on the system to determine whether the system is behaving as expected or if an error has occurred. Because the computer's memory is finite, the bits used to

store the measurement results utilized in error-correction must eventually be erased to make way for new measurement results. According to Landauer's principle, this erasure carries an associated energy cost that must be accounted for when tallying the total energy of the computation. We analyze the energy cost associated with error correction in more detail in later chapters.

What can we conclude from our study of reversible computation? There are three key ideas. First, reversibility stems from keeping track of every bit of information; irreversibility occurs only when information is lost or erased. Second, by doing computation reversibly, we obviate the need for energy expenditure during computation. All computation can be done efficiently, without production of garbage bits whose value depends upon the input to the computation. That is, if there is an irreversible circuit computation a function f , then there is an efficient simulation of this circuit by a reversible circuit with the action $(x, y) \rightarrow (x, y \oplus f(x))$.

What are the implication of these result for physics, computer science, and for quantum computation and quantum information? From the point of view of a physicist or hardware engineer worried about head dissipation, the good news is that, in principle, it is possible to make computation dissipation -free by making it reversible, although in practice energy dissipation is required for system stability and immunity from noise. At an ever more fundamental level, the ideas leading to reversible computation also lead to the resolution of a century-old problem in the foundations of physics, the famous problem of Maxwell's demon. From the point of view of a computer scientist, reversible computation validates the use of irreversible models of computation such as the Turing Machine (since using them or not gives polynomially equivalent models). Moreover, since the physical world is fundamentally reversible, one can argue that complexity classes based upon reversible models of computation are more natural than complexity classes based upon irreversible models of computation. From the point of view of quantum computation and quantum information, reversible computation is enormously important. To harness the full power of quantum computation, any classical subroutines in a quantum computation must be performed reversibly and without the production of garbage bits depending on the classical input.

Example 4.4.4. (From Fredkin to Toffoli and back again) What is the smallest number of Fredkin gates needed to simulate a Toffoli gate? What is the smallest number of Toffoli gates needed to simulate a Fredkin gate?

Important Note

Maxwell's Demon

The laws of thermodynamics govern the amount of work that can be performed by a physical system at thermodynamic equilibrium. One of these laws, the second law of thermodynamics, states that the entropy in a closed system can never decrease. In 1871, James Clerk Maxwell proposed the existence of a machine that apparently violated this law. He envisioned a miniature little ‘demon’, like that shown in the figure below, which could reduce the entropy of a gas cylinder initially at equilibrium by individually separating the fast and slow molecules into the two halves of the cylinder. This demon would sit at a little door in the middle partition. When a fast molecule approaches from the left side the demon opens a door between the partitions, allowing the molecule through, and then closes the door. By doing this many times the total entropy of the cylinder can be decreased, in apparent violation of thermodynamics.

The resolution of the Maxwell’s demon paradox lies in the fact that the demon must perform measurements on the molecules moving between the partitions, in order to determine their velocities. The result of this measurement must be stored in demon’s memory. Because any memory is finite, the demon must eventually begin erasing information from its memory, in order to have space for new measurement results. By Landauer’s principle, this act of erasing information increases the total entropy of the combined system - demon, gas cylinder, and their environments. In fact, a complete analysis shows that Landauer’s principle implies that the entropy of the combined system is increased at least as much by this act of erasing information as the entropy of the combined system is decreased by the actions of the demon, thus ensuring that the second law of thermodynamics is obeyed.

This section constructs, for any classical computation, a quantum circuit that can perform the same computation with comparable efficiency. This result proves that quantum computation is at least as powerful as classical computation. In addition, many quantum algorithms begin by using this construction to compute a classical function on a superposition of values prior to using nonclassical means for efficiently extracting information from this superposition.

The construction of quantum analogs to all classical computations relies on a classical result that constructs a reversible analog to any classical computation as shown in theorem 6.4.1 where we described the relations between classical reversible computation and both general classical computation and quantum computation. We

now show reversible versions of Boolean logic gates and quantum analogs of these reversible versions. Given a classical reversible circuit composed of reversible Boolean logic gates, simple substitutions of the analogous quantum gates for the reversible gates gives the desired quantum circuit. The hard step in proving that every classical computation has comparably efficient quantum analog is proving that every classical computation has a reversible version of comparable efficiency. Although this construction is purely classical, it is of such fundamental importance to quantum computation that we present it here. We then provide this construction and then describe the language that is used to specify explicit quantum circuit for several classical functions such as arithmetic operations.

4.4.2 From Reversible classical computations to quantum computations

Any sequence of quantum transforms effects a unitary transformation U on the quantum systems. As long as no measurements are made, the initial quantum state of the system prior to a computation can be recovered from the final quantum state $|\psi\rangle$ by running $U^{-1} = U^\dagger$ on $|\psi\rangle$. Thus, any quantum computation is reversible prior to measurement in the sense that the input can always be computed from the output.

In contrast, classical computations are not in general reversible: it is not usually possible to compute the input from the output. For example, while the classical NOT operation is reversible, the AND, OR, and NAND are not. Every classical computation does, however, have a classical reversible analog that takes only slightly more computational resources. We now show how to make basic Boolean gates reversible. Then show how to make entire Boolean circuits reversible in a resource efficient way, considering space, the number of qubits required, and the number of primitive gates. This construction of efficient classical reversible versions of arbitrary Boolean circuits easily generalizes to a construction of quantum circuits that efficiently implement general classical circuits.

Any classical reversible computation with n input and n output bits simply permutes the $N = 2^n$ bit strings. Thus, for any such classical reversible computation there is a permutation $\pi : Z_N \rightarrow Z_N$ sending an input bit string to its output bit string. This permutation can be used to define a quantum transformation.

$$U_\pi : \sum_{x=0}^{N-1} a_x |x\rangle \rightarrow \sum_{x=0|\pi(x)\rangle}$$

that behaves on the standard basis vectors, viewed as classical bit strings, exactly as

π did. The transformation U_π is unitary, since it simply reorders the standard basis elements.

Any classical computation on n input and m output bits defines function

$$f : Z_N \rightarrow Z_M$$

mapping the $N = 2^n$ input bit strings to the $M = 2^m$ output bit strings. Such a function can be extended in a canonical way to a reversible function π_f acting on $n + m$ bits partitioned into two registers, the n -bit input register and the m -bit output register:

$$\begin{aligned} \pi_f : Z_L &\rightarrow Z_L \\ (x, y) &\rightarrow |(x, y \oplus f(x))\rangle \end{aligned}$$

where \oplus denotes the bitwise exclusive-OR. The function π_f acts on the $L = 2^{n+m}$ bit strings, each made up of an n -bit string x and an m -bit string y . For $y = 0$, the function pi acts like f , except that the output appears in the output register and the input register retains the input. There are many other ways of making a classical computation reversible, and for a particular classical computation, there may be a reversible version that requires fewer bits, but this construction always works.

Since π_f is reversible, there is a corresponding unitary transformation $U_f : |x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$. Graphically the transformation U_f is depicted as ??.

We have already shown how to implement any unitary operation in terms of simple gates. For most unitary transformations, that implementation is highly inefficient. While most unitary operators do not have an efficient implementation, U_f has an efficient implementation as long as there is a classical circuit that computes f efficiently. The method for constructing an efficient implementation of U_f from an efficient classical circuit for f has two parts. The first part constructs an efficient reversible classical circuit that computes f . The second part substitutes quantum gates for each of the reversible gates that make up the reversible classical circuit. We thus define reversible Boolean logic gates and convert the easy second part of the construction. We then explain the involved construction of an efficient reversible classical circuit for any efficient classical circuit.

4.4.3 reversible and Quantum versions of simple classical gates

This section describes reversible versions of the Boolean logic gates NOT, XOR, AND, and NAND. Quantum versions of these gates act like the reversible gates on elements of the standard basis. Their action on other input states is prescribed by

the linearity of quantum operations; the action of a gate on a superposition is the linear combination of the action of the gate on the standard basis elements making up the superposition. In this way, the behavior of a reversible gate fully defines the behavior of its quantum analog, and vice versa. The tight connection between the two allows us to use the same notation for both the gates with the understanding that the quantum gates can be applied to arbitrary superpositions, whereas the classical reversible gates are applied to bit strings that correspond to the standard basis elements.

Let b_1 and b_0 be two binary variables, variables taking on only values 0 or 1. We define the following quantum gates:

NOT

The NOT gate is already reversible. We will use X to refer to both the classical reversible gate and the single-qubit operator $X = |0\rangle\langle 1| + |1\rangle\langle 0|$, which performs a classical NOT operation on classical bits encoded as the standard basis elements.

XOR

The controlled negation performed by the CNOT gate amounts to an *XOR* operation on its input values. It retains the value of the first bit b_1 , and replaces the value of the bit b_0 with the XOR of the two values.

The quantum version behaves like the reversible version on the standard basis vectors, and its behavior on all other states can be deduced from the linearity of the operator.

AND

It is impossible to perform a reversible AND operation with only two bits. The three-bit controlled-controlled-NOT gate, or Toffoli gate, can be used to perform a reversible AND operation.

$$CCNOT |b_1 b_0, 0\rangle = |b_1 b_0, b_1 \wedge b_0\rangle$$

where \wedge is notation for the classical AND of the two bit values.

The Toffoli gate is defined for all input: when the value of the third bit is 1,

$$CCNOT |b_1 b_0, 1\rangle = |b_1 b_0, 1 \oplus b_1 \wedge b_0\rangle$$

By varying the values of input bits, the Toffoli gate T can be used to construct a complete set of Boolean connectives, not just the classical AND. Thus, any combinatorial circuit can be constructed from Toffoli gates alone. The Toffoli gate computes NOT, AND, XOR, and NAND in the following way:

$$\begin{aligned} CCNOT |1, 1, x\rangle &= |1, 1, \neg x\rangle \\ CCNOT |x, y, 0\rangle &= |x, y, x \wedge y\rangle \\ CCNOT |1, x, y\rangle &= |1, x, x \oplus y\rangle \\ CCNOT |x, y, 1\rangle &= |x, y, \neg(x \wedge y)\rangle \end{aligned}$$

where \neg indicates the classical NOT acting on the bit value.

An alternative to the Toffoli gate, the Fredkin gate F , acts as a controlled swap. The Fredkin gate F , like the Toffoli gate, can implement a complete set of classical Boolean operators:

$$\begin{aligned} F |x, 0, 1\rangle &= |x, x, \neg x\rangle \\ F |x, y, 1\rangle &= |x, y \vee x, y \vee \neg x\rangle \\ F |x, 0, y\rangle &= |x, y \wedge x, y \vee \neg x\rangle \end{aligned}$$

where \vee is the notation for the classical OR of the two bit values.

Because a complete set of classical Boolean connectives can be implemented using just the Toffoli gate, or the Fredkin gate F , these gates can be combined to realize arbitrary Boolean circuits. We now describe the explicit implementation of certain classical functions. As the equations for the Toffoli gate illustrate, the operations CNOT and X can be implemented by Toffoli gate with the addition of one or two bits permanently set to 1. For clarity, we use CNOT and X gates in our construction, but all constructions can be done using only Toffoli gates, since we can replace all uses of CNOT and X with Toffoli gates that have additional input bits with their input value set appropriately. For example, the circuit shown in figure ?? implements a one-bit full adder using Toffoli and controlled-NOT gates, where x and y are the data bits, s is their sum (modulo 2), c is the incoming carry bit, and c' is the new carry bit. Several one-bit adders can be strung together to achieve full n -bit addition.

4.5 Reversible implementation of Classical Circuits

This section develops systematic way to turn arbitrary classical Boolean circuits into reversible classical circuits of comparable computational efficiency in terms of

the number of bits and the number of gates. The resulting reversible circuits are composed entirely of Toffoli and negation gates. A quantum circuit with the same efficiency as the classical reversible circuit is obtained by the trivial substitution of quantum Toffoli and X gates for classical Toffoli and negation gates. Thus, as soon as we have an efficient version of a computation in terms of Toffoli gates, we immediately know how to obtain a quantum implementation of the same efficiency.

4.5.1 A Naive reversible Implementation

Rather than start with arbitrary Boolean circuits, we consider a classical machine that consists of a register of bits and a processing unit. The processing unit performs simple Boolean operations or gates on one or two of the bits in the register at a time and stores the result in one of the register's bits. We assume that, for a given size input, the sequence of operations and their order of execution are fixed and do not depend on the input data or on other external control. In analogy, with quantum circuits, we draw bits of the register as horizontal lines. A simple program (for four-bit conjunction) or this kind of machine is depicted in figure ??.

An arbitrary Boolean circuit can be transformed into a sequence of operations on a large enough register to hold input, output and intermediate bits. The space complexity of a circuit is the size of the register.

Computations performed by this machine are not reversible in general; by reusing bits in the register, the machine erases information that cannot be reconstructed later. A trivial, but highly space inefficient, solution to this problem is not to reuse bits during the entire computation. Figure ?? illustrates how the circuit can be made reversible by assigning the results of each operation to a new bit. The operation that reversibly computes the conjunction and leaves the result in a bit initially set to 0 is, of course, the Toffoli gate. Since the NOT gate is reversible, and NOT together with AND form a complete set of Boolean operations, this construction can be generalized to turn any computation using Boolean logic operations into one using only reversible gates. This implementation, however, needs an additional bit for every AND performed, so if the original computation takes t steps, then a reversible one constructed in this naive way requires up to t additional bits of space.

Furthermore, this additional space is no longer in the 0 state and cannot be directly reused, for example, to compose two reversible circuits. Reusing temporary bits will be crucial to keeping the space requirements close to that of the original nonreversible classical computation. Resetting a bit to zero is not as trivial as it might seem. A transformation that reset a bit to 0, regardless of whether it was 0 or 1 before, is not reversible (it loses information), so it cannot be used as part

of reversible computation. Reversible computations cannot reclaim space through a simple reset operation. They can, however uncompute any bit set during the course of a reversible computation by reversing the part of the computation that computed the bit.

Example 4.5.1. Consider the computation of figure ???. Bits t_1 and t_0 are temporarily used to obtain the output in bit m_0 . Figure ?? shows how to uncompute these bits, resetting them to their original 0 value by reversing all but the last step of the circuit in figure ??, so that they may be reused as part of a continuing computation. Here the temporary bits are reclaimed at the cost of roughly doubling the number of steps.

We can reduce the number of qubits needed by uncomputing them and reusing them in the course of the algorithm. The method of uncomputing bits by eprforming all of these steps in reverse order, except those giving the output, works for any classical Boolean subcircuit. The naive construction requires up to t additional bits in the register.

Example 4.5.2. Suppose we want to construct the conjunction of eight bits. Simply reversing the steps, generalizing the approach shown in figure ??, would require six additional temporary bits and one bit for the final output. We can save space by using the four-bit AND circuit of figure ?? and then combining the results as shown in figure ???. This construction uses two temporary bits in addition to the two temporary bits used in each of the four-bit ANDs. Since each of the four-bit ANDs uncomputes its temporary bits, these bits can be reused by the t four-bit ANDs. This circuit uses only a total of four additional temporary bits, though it does require more gates.

There is an art to deciding when to uncompute which bits to maintain efficiency and to retain subresults in the computation. The key ideas of this section, adding bits to obtain reversibility and uncomputing their values so that they may be reused, are the amin ingredients of the general construction. By choosing carefully when and what to uncompute, it is possible to make a positive tradeoff, sacrificing some additional gates to obtain a much more efficient use of space. Examples, such as an explicit implementation of an m -way AND.

4.5.2 A General Construction

This section shows how, by carefully choosing which bits to uncompute when, a reversible version of any classical computation can be achieved with only minor

increases in the number of gates. We show that any classical circuit using t gates and s bits, has a reversible counterpart using only $O(t^{1+\epsilon})$ gates and $O(s \log t)$ bits. For $t \gg s$, this construction uses significantly less space than the $(s + t)$ space of the naive approach described earlier at only a small increase in the number of gates.

In order to understand how to obtain these bounds, we must consider carefully how many bits are being used in what way. Let C be a classical circuit, composed of AND and NOT gates, that uses not more than t gates and s bits. The circuit C can be partitioned into $r = \lceil t/s \rceil$ subcircuits each containing s or fewer consecutive gates $C = C_1 C_2 \dots C_r$. Each subcircuit C_i has s input and s output bits, some of which may be unchanged.

Using techniques from the previous section, each circuit C_i can be replaced by a reversible circuit R_i that uses at most s additional bits as shown in figure ???. The circuit R_i returns its input as well as the s output values in the subsequent computation. The input values will be used to compute and uncompute R_i in order to save space.

More than s gates may be required to construct R_i . In general, R_i can be constructed using at most $3s$ gates. While other more efficient constructions are possible, the following three steps always work.

4.5.3 Reversible Computation

In order to implement Shor's algorithm, it is necessary to efficiently perform some of the arithmetic operations described above (modular exponentiation, in particular) with a quantum compute. Intuitively this seems like it should not be a problem, given that there is an efficient way to implement these operations classically. But does an efficient classical implementation necessarily imply the existence of an efficient quantum implementation? The fact that classical algorithms can use non-unitary operations such as ANDs and ORs, while quantum computers are restricted to unitary operations, makes this question non-trivial to answer.

Before we address the question of whether efficient classical algorithms for arithmetic problems (or any other type of computational problem) imply efficient quantum algorithms for these problems, we need to briefly discuss the notion of quantum circuit complexity. What does it mean to have an efficient quantum algorithm for some task?

Our answer to this question will be similar to the classical case. In order to completely specify a quantum algorithm, it is necessary to give a construct that produces a quantum circuit solving the problem at hand for any given input size. Also, as in the classical case, one assume the quantum circuit is composed entirely

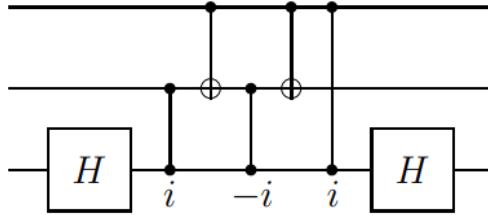
of quantum gates acting on small number of qubits-in analogy to the classical case, one may require that all gates in the quantum circuit act on just one or two qubits. Again, to be considered efficient, one usually requires that the number of gates is polynomially related to the number of input qubits.

There is, however, a crucial difference between the classical and quantum cases. Whereas there are a finite number of classical one and two bit operations, and it is well known that AND and NOT operations (for instance) are sufficient to generate any such operation, there are infinitely many one and two qubit operations. Although there are interesting technical aspects of this issue, such as the degree to which a finite number of quantum gates can approximate arbitrary gates, we will essentially sweep the issue under the rug by saying that arbitrary one and two qubit gates are allowed in our circuits.

By the way, it will be very helpful to use Toffoli gates to construct quantum circuits for various tasks. Recall that a Toffoli gate performs the transformation

$$T |a\rangle |b\rangle |c\rangle = |a\rangle |b\rangle |c \oplus (a \wedge b)\rangle$$

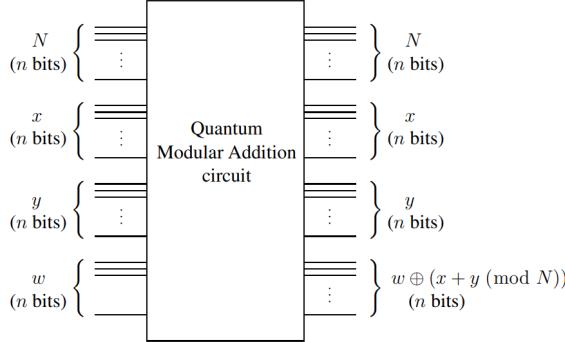
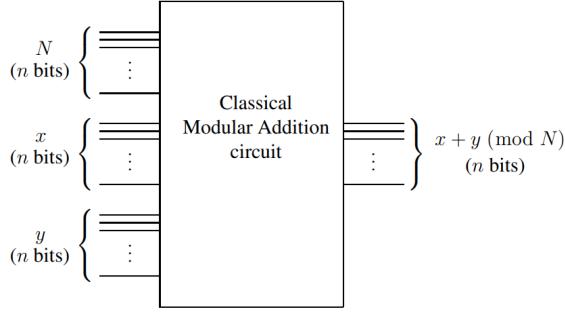
for $a, b, c \in \{0, 1\}$. A Toffoli gate can be decomposed into one- and two-qubit quantum gates as follows: A careful check of the 8 standard basis states show that the circuit



implements a Toffoli gate as claimed.

Now let us return to the question of whether an arbitrary efficient classical algorithm can be converted to an efficient quantum algorithm. Suppose that we have some classical Boolean circuit. For sake of example, suppose it performs modular additions as suggested in the following diagram. Obviously, such a circuit is non-unitary-the number of input and output qubits do not even agree. If we are to somehow transform this circuit into a valid quantum circuit, it will have to correspond to a unitary transformation.

Based on our previous discussions concerning black-box transformations, we might hope to implement this operation unitarily as follows: This is a special type of unitary transformation, because it always maps classical states to classical states, therefore



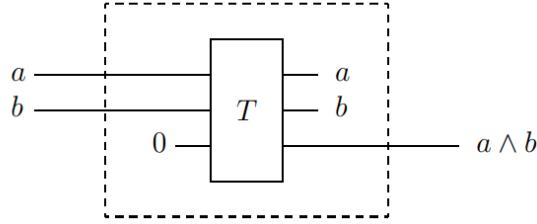
giving rise to a permutation matrix. We will show how to implement such transformations using only gates that map classical states to classical states, so none of what we are discussing right now needs to be thought of as inherently quantum. Usually when we are talking about quantum gates and circuits that always map classical states to classical states, we refer to them as reversible gates or circuits. In other words, a reversible gate is a special type of quantum gate that maps classical states to classical states, or equivalently gives rise to a permutation matrix.

Now let us see how we can transform the original classical circuit into such a circuit, obeying the constraint that only reversible gates are used inside the circuit. We do not want to have to reinvent the wheel, so to speak, so we can begin by trying to replace the gates of the classical circuit with reversible gates. The Toffoli gates will be handy for doing this.

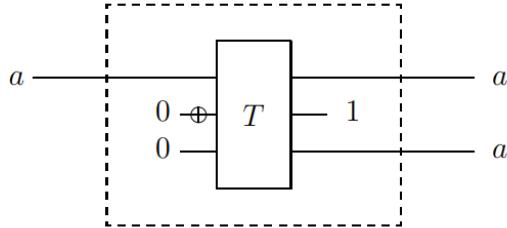
AND gates. We can simulate AND gate using a toffoli gate as follows: **NOT gates** These are already unitary, so nothing needs to be done.

OR gates These can be replaced by an AND gate and three NOT gates using DeMorgan's Laws.

FANOUT We often do not even think of this as an operation, but we could have fanout in a classical Boolean circuit, and this needs to be explicitly implemented in

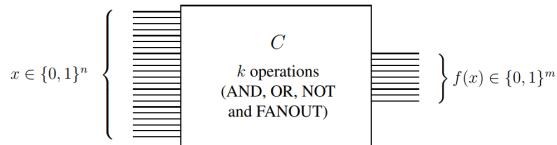


a quantum circuit. note that this does not mean replicating the quantum state of a qubit - it means replicating the classical state. here is how this operation can be simulated using a Toffoli gate:

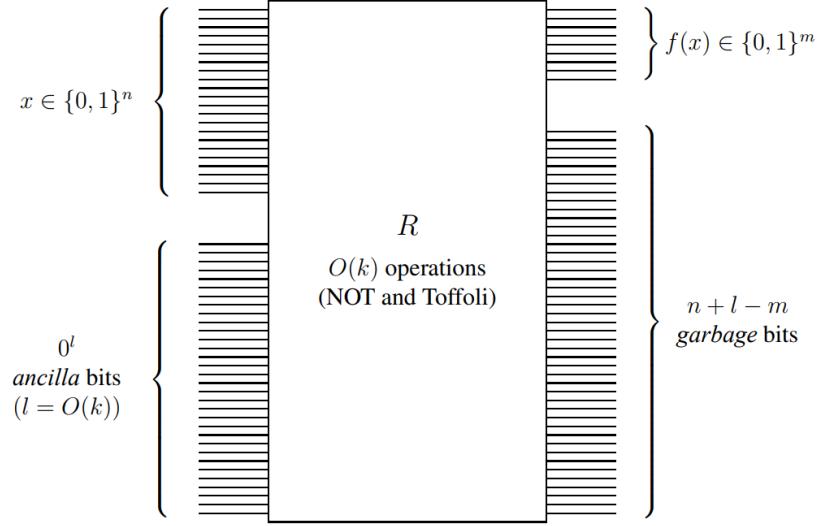


Of course it would have been just as easy to input a 1 into the second input of the Toffoli gate rather than performing a NOT operation on a 0. I've only written it like this because it will be convenient later to have all of the auxiliary inputs start in the 0 state.

Now, we can simply go through the original circuit and replace each of the AND, OR, NOT, and FANOUT operations with reversible gates as just described,. if we started with a general circuit C computing a function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ as follows: Then we would end up with a reversible circuit R as follows.



The auxiliary bits needed for the AND and FANOUT simulations in this circuit are sometime called ancilla bits. The bits labeled garbage are the extra bits that resulted from the simulations (that were left inside the dashed boxes in the above pictures).



The circuit above is not quite what we want, because the garbage bits are undesirable - we have no control over them, and they would cause problems in many quantum computations. However, we almost have what we are after. By taking advantage of the fact that NOT gate and Toffoli gate are their own inverses, we can simply take a “mirror image” of R to get R^{-1} , which simply undoes the computation of R . Combining R, R^{-1} , and a collection of controlled-NOT gates as described in figure essentially gives us what we are looking for. The only difference between this circuit and our initial aim is the existence of ancilla bits. (It should be noted that commonly when people use the term “ancilla”, they often expect that such bits are returned to their initial 0 state as in the circuit in Figure 1.) These bits will not have any adverse effects, and often we do not even explicitly mention them.

So, in summary, if you have a classical Boolean circuit C for computing some function

$$f : \{0,1\}^n \rightarrow \{0,1\}^m$$

then the procedure above will transform a description of this circuit into a description of a quantum circuit S_C composed only of Toffoli gates, NOT gates and controlled-NOT gates that satisfies

$$S_C |x\rangle |y\rangle |0^l\rangle = |x\rangle |y \oplus f(x)\rangle |0^l\rangle$$

The number of ancilla qubits l and the total number of gates in S_C is linear in the number of operations in the original circuit C . The presence of ancilla qubits will have no effect on any algorithm that uses the circuit S_C , and in fact they could be

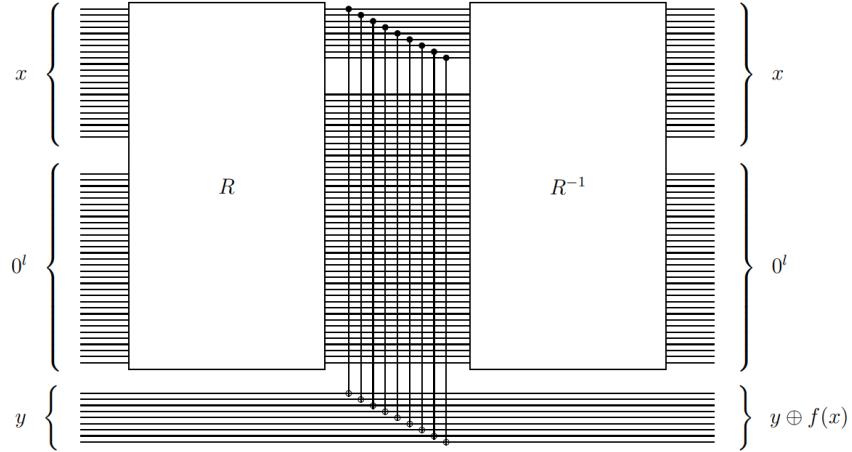


Figure 4.17: The final reversible circuit S_C implementing the transformation we are after

re-used to perform some other transformation after S_C is performed. For this reason, we will usually say that S_C performs the transformation

$$S_C |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$$

leaving it implicit that some number of ancilla qubits may have been used to implement S_C .

4.5.4 Reversible Implementation of invertible functions

Finally, suppose that you have a function of the form

$$f : \{0, 1\}^n \rightarrow \{0, 1\}^n$$

that itself is invertible (one-to-one and onto), and suppose that f is efficiently computed by some Boolean circuit C . We already know that it is possible to build a reversible S_C that performs the transformation

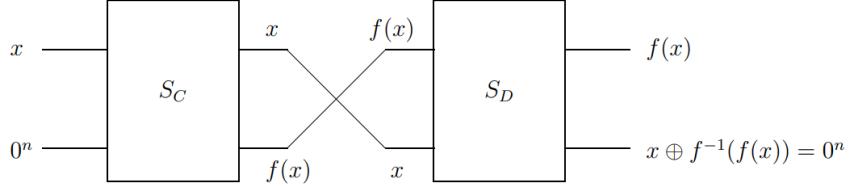
$$S_C |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$$

for all $x, y \in \{0, 1\}^n$ (possibly using some number of ancilla bits). Is it possible to go further and build a reversible circuit Q , possibly using some ancilla bits, that efficiently performs the transformation

$$Q |x\rangle = |f(x)\rangle$$

for all $x \in \{0, 1\}^n$? because f is assumed to be invertible, the transformation to be performed by Q is indeed reversible, so in principle it is possible to build Q . The question, however, is about efficiency.

Without any further assumption it is not known how to construct Q given only C . However, if in addition we have a Boolean circuit D that efficiently computes f^{-1} , then Q can be constructed that efficiently performs the required transformation as follows: (In order to simplify the picture, each “wire” represents n bits and the



ancilla bits needed by the circuits S_C and S_D have not been shown explicitly.

4.6 Complexity of Quantum Algorithms

Definition 4.6.1. Efficiency of Quantum Algorithm: Let n be the number of input bits. A Quantum Algorithm is efficient if the number of gates in the quantum circuit is $\mathcal{O}(\text{poly}(n))$.

Definition 4.6.2. Probabilistic nature of quantum algorithms: 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.

Definition 4.6.3. (Computational Cost/Query Complexity): In Quantum Algorithms, Computational cost is equal to the Query Complexity. The query complexity is the number of times we call the query oracle (U_f).

The goal is to 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 4.6.1. (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 4.6.2. (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.

Definition 4.6.4. (Gate Complexity): The number of *elementary gates* needed to implement the oracle U_f .

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.**

Definition 4.6.5. (Circuit Depth): 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.

Definition 4.6.6. (Coherence Time): Quantum states can be preserved only for a short amount of time called coherence time.

Thus circuit depth can be used to check whether the coherence time is exceeded by a Quantum Computer.

Quantum Algorithms should be depth efficient. Thus, try to reduce the circuit time even if we need to run it multiple times.

A Quantum Algorithm consists 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.

Definition 4.6.7. (Working Register): 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.

Note that we do not factor the cost of working registers into asymptotic cost analysis in the literature.

4.7 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 4.18. 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

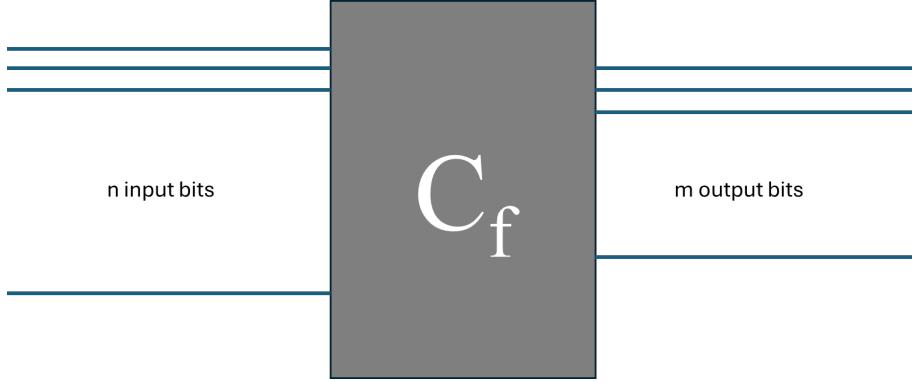


Figure 4.18: Classical Circuit

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 set 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. It's

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 3.4.2). 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 4.19 Thus, if $x = 1$ then the state of the system is swapped i.e.

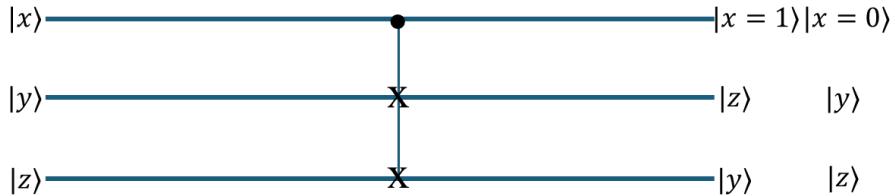


Figure 4.19: Control Swap Gate

$|y\rangle$ 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 4.20. The truth table for the above circuit is as shown in the

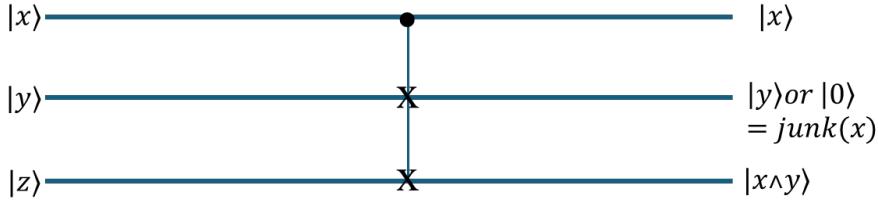


Figure 4.20: Reversible AND Gate

table 4.1. Hence, we can clearly see that the output of the circuit is the same as the

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 4.1: Reversible AND Gate

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

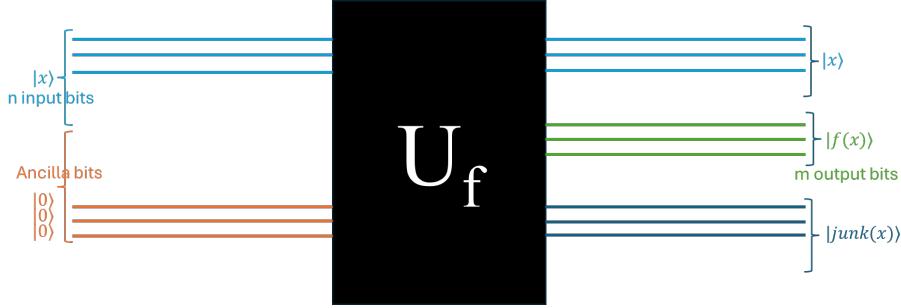


Figure 4.21: Quantum Circuit

inputs equal to the number of outputs, produces the same output as the classical function f along with the input and some junk values as shown in the figure 4.21. **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 4.21, 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 4.22 Using this circuit it can be seen that the output of the entire circuit is

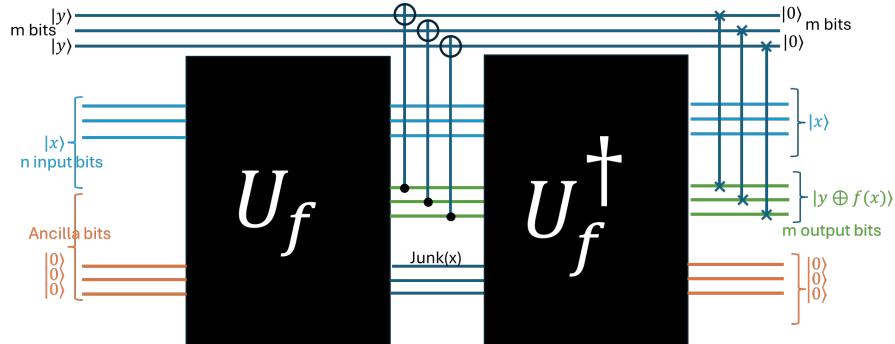


Figure 4.22: Quantum Oracle

$|x\rangle$, $|y \oplus f(x)\rangle$ 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 4.23.

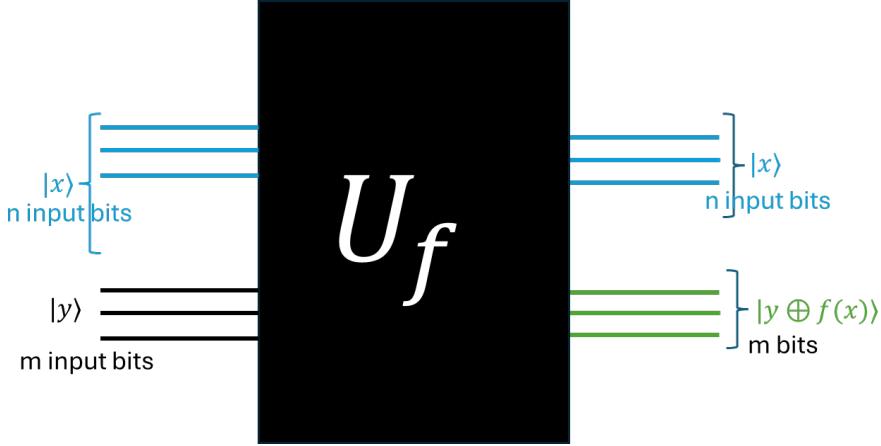


Figure 4.23: Oracle

4.8 Quantum Algorithms as Quantum Circuits

Now let's consider quantum circuits as shown in the figure 4.24. 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.

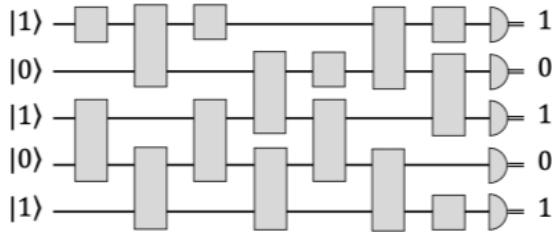


Figure 4.24: Generic form of a Quantum Circuit

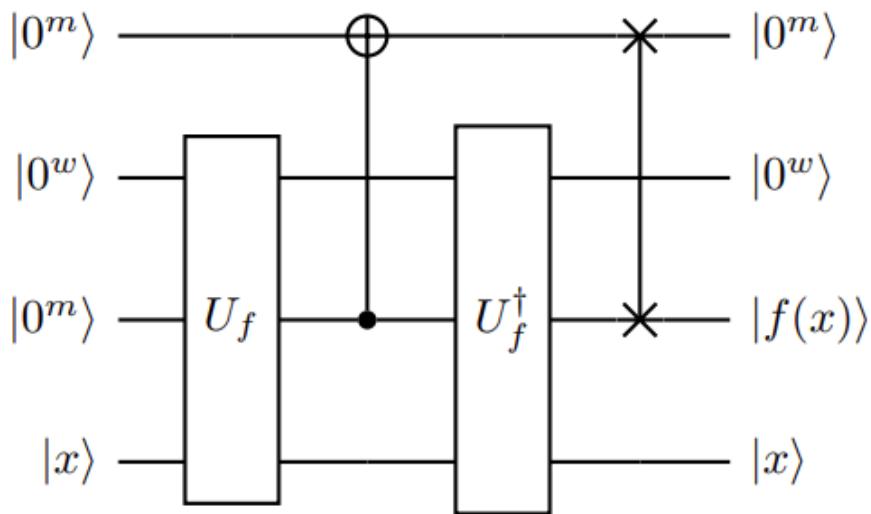


Figure 4.25: Uncomputation of a classical gate

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.

4.9 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.

Theorem 4.9.1. *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.*

Chapter 5

Number-Theoretic Problems

We now understand the topic with which it is concerned: The computational number theory.

5.1 Arithmetic Problems

5.1.1 Integer Addition

Input: $x, y \in \mathbb{Z}$

Output: $x + y$

One can formalize the notion for an efficient algorithm for a given problem in various ways by describing precise models of computation. For the purposes of this discussion let us just think in terms of Boolean circuits and identify an algorithm with a collection of acyclic circuits (one for each possible input length) that produce the correct outputs for all possible input strings. The circuits should be composed of *AND*, *OR*, and *NOT* gates, along with FANOUT operations (which we would often not consider as gates at all), and in order to be considered efficient, the total number of gates in each circuit should be polynomial in the number of input bits.

The number of bits needed to write down the number x in binary notation is

$$lg(x) = \begin{cases} 1 & \text{if } x = 0 \\ \lfloor \log_2 |x| \rfloor + 2 & \text{if } x \neq 0 \end{cases}$$

assuming we keep a sign bit for each non-zero integer, and so the number of input bits of the above problem is $\log(x) + \log(y)$. Now, if you consider the usual method

for adding number that you probably learned in elementary school or before, but using base 2 instead of base 10, you could turn that method into a construction of Boolean circuits of size $O(\log_2(x) + \log_2(y))$ for solving this problem. This is in fact linear in the input size, and so this algorithm will be considered to be very efficient.

By the way, I should mention that it is typical to place additional constraints on collections of Boolean circuits in order for the circuits to be identified with efficient algorithms. In particular, the circuits should satisfy certain uniformity constraints, which means that not only do they have polynomial size, but moreover there should not exist an efficient construction of the circuits themselves. This issue can be ignored in this class, however.

Summing up, we would simply say that the Boolean circuit complexity, or “bit complexity” of computing $x + y$ for integers x and y is $\mathcal{O}(\log_2 x + \log_2 y)$. Or even more simply, the bit complexity of adding two n bit integers is $\mathcal{O}(n)$. Here are some other arithmetic and number theoretic problems:

5.1.2 Integer Multiplication

Input: $x, y \in \mathbb{Z}$

Output: xy

The “elementary school” multiplication algorithm established that the bit complexity of multiplying x and y is $\mathcal{O}((\lg x)(\lg y))$, or more simply that the bit complexity of multiplying two n bit numbers is $\mathcal{O}(n^2)$. In fact, there are asymptotically better methods, such as the Schonhage-Strassen Algorithm that has bit complexity $\mathcal{O}(n \log n \log \log n)$. (Large constant factors in the running time make this particular method less efficient than other methods until n becomes quite large - several thousand bits perhaps.)

5.1.3 Integer Division

Input: Integers x and $y \neq 0$

Output: Integers a and b , with $0 \leq |b| < |y|$ such that $x = ay + b$

The cost is roughly the same as integer multiplication - the “elementary school” division method gives an algorithm with bit complexity $\mathcal{O}((\log_2 x)(\log_2 y))$. (in fact it is actually somewhat better: $\mathcal{O}((\log_2 a)(\log_2 b))$.)

5.1.4 Greatest Common Divisor

Input: Non-negative integers x and y

Output: $\gcd(x, y)$

Euclid's algorithm, which is over 2000 years old, computes $\gcd(x, y)$ in $O((\log_2 x)(\log_2 y))$ bit operations.

5.1.5 Modular Integer Addition

Input: A positive integer N and integers $x, y \in \mathbb{Z}_N = \{0, \dots, N - 1\}$

Output: $x + y \bmod N$

The bit complexity of this problem is $O(\log_2 N)$.

5.1.6 Modular integer Multiplication

Input: A positive integer N and integers $x, y \in \mathbb{Z}_N$

Output: $xy \pmod{N}$

Here the bit complexity is $O((\log_2 N)^2)$ by the elementary school algorithm, again asymptotically faster methods exist.

5.1.7 Modular Inverse

Input: A positive integer N and an integer $x \in \mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$.

Output: $y \in \mathbb{Z}_N$ such that $xy \equiv 1 \pmod{N}$

The bit complexity is $O((\log_2 N)^2)$.

5.1.8 Modular Exponentiation

Input: A positive integer N , an integer $x \in \{0, \dots, N - 1\}$, and any integer k .

Output: $x^k \pmod{N}$

Using the method of repeated squaring, the bit complexity is $O((\log_2 k)(\log_2 N)^2)$.

5.1.9 Primality testing

Input: A positive integer N

Output: "prime" if N is a prime number, "not prime" otherwise.

Randomized algorithms having bit complexity $O((\log_2 N)^3)$ that allow a probability 1/4, say, of making an error have been known for many years. In a breakthrough a few years ago, Agarwal, Kayal and Saxena gave a deterministic prime testing algorithm that gives a $O((\log_2 N)^6)$ deterministic algorithm for testing primality.

5.1.10 Integer Factoring

Input: A positive integer N

Output: A prime factorization $N = P_1^{a_1} \dots p_k^{a_k}$ of N

No classical algorithm is known to give a polynomial bit complexity of factoring
- the best known algorithm asymptotically gives a bit complexity of

$$2^{O((\log N)^{1/3}(\log \log N)^{2/3})}$$

We will see that when we turn to quantum algorithms, Shor's algorithm will solve the Integer Factoring problem using $O((\log_2 N)^3)$ operations, giving a remarkable speed-up over known classical algorithms for this task.

Chapter 6

Entanglement

What are the allowable quantum states of systems of several particles? The answer to this is enshrined in the addendum of the first postulate of quantum mechanics: the superposition principle. In this chapter we will consider a special case, system for two qubits. This chapter will facilitate an intuitive understanding of the phenomenon of quantum entanglement - phenomenon which is responsible for much of the “quantum weirdness” that makes quantum mechanics so counter-intuitive and fascinating.

6.1 Two qubits

Now let us examine a system of two qubits. Consider the two electrons in the hydrogen atom, each can be regarded as a 2-state quantum system: Since each electron can be either of the ground state or the excited state, classically the two electrons are in one of the four states - 00, 01, 10 or 11 - and represent 2 bits of classical information. By the superposition principle, the quantum state of the two electrons can be any linear combination of these four classical states:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$

where $\alpha_{ij} \leq \mathbb{C}$, $\sum_{ij} |\alpha_{ij}|^2 = 1$. This is just the dirac notation for the unit vector in \mathbb{C}^4 :

$$\begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix}$$

Measurement:

As in the case of a single qubit, even though the state of two qubits is specified by

four complex numbers, most of this information is not accessible by measurement. In fact, a measurement of a two qubit system can only reveal two bits of information. The probability that the outcome of the measurement is two bit string $x \in \{0, 1\}^2$ is $|\alpha_x|^2$. Moreover, following the measurement the state of the two qubits is $|x\rangle$. i.e. if the first bit of x is j and the second bit k , then following the measurement, the state of the first qubit is $|j\rangle$ and the state of the second is $|k\rangle$. An interesting question comes up here: what if we measure just the first qubit? What is the probability that the outcome is 0? This is simple. It is exactly the same as it would have been if we had measured both qubits: $Pr\{1\text{st bit}=0\} = Pr\{00\} + Pr\{01\} = |\alpha_{00}|^2 + |\alpha_{01}|^2$. But how does this partial measurement disturb the state of the system? The answer is obtained by an elegant generalization of our previous rule for obtaining the new state after a measurement. The new superposition is obtained by crossing out all those terms of $|\psi\rangle$ that are inconsistent with the outcome of the measurement (i.e. those whose first bit is 1). Of course, the sum of the squared amplitudes is no longer 1, so we must renormalize to obtain a unit vector:

$$|\phi_{new}\rangle = \frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}$$

Entanglement:

Suppose the first qubit is in the state $3/5|0\rangle + 4/5|1\rangle$ and the second qubit is in the state $1/\sqrt{2}|0\rangle - 1/\sqrt{2}|1\rangle$, then the joint state of the two qubits is $(3/5|0\rangle + 4/5|1\rangle)(1/\sqrt{2}|0\rangle - 1/\sqrt{2}|1\rangle) = 3/5\sqrt{2}|00\rangle - 3/5\sqrt{2}|01\rangle + 4/5\sqrt{2}|10\rangle - 4/5\sqrt{2}|11\rangle$.

More generally, if the state of the first qubit is $\alpha_0|0\rangle + \alpha_1|1\rangle$ and the state of the second qubit is $\beta_0|0\rangle + \beta_1|1\rangle$, then the joint state of the two qubits is $\alpha_0\beta_0|00\rangle + \alpha_0\beta_1|01\rangle + \alpha_1\beta_0|10\rangle + \alpha_1\beta_1|11\rangle$. Can every state of two qubits be decomposed in this way? Our classical intuition would suggest that the answer is obviously affirmative. After all each of the two qubits must be in some state $\alpha|0\rangle + \beta|1\rangle$, and so the state of the two qubits must be the product. In fact, there are states such as $|\phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ which cannot be decomposed in this way as a state of the first qubit and that of the second qubit. Can you see why? Such a state is called an entangled state. When the two qubits are entangled, we cannot determine the state of each qubit separately. The state of the qubits has as much to do with the relationship of the two qubits as it does with their individual states. If the first (resp. second) qubit of $|\phi^+\rangle$ is measured then the outcome is 0 with probability 1/2 and 1 with probability 1/2. However if the outcome is 0, then a measurement of the second qubit results in 0 with certainty. This is true no matter how large the spatial separation between the two particles.

The state $|\Phi^+\rangle$, which is one of the Bell basis states, has a property which is even

more strange and wonderful. The particular correlation between the measurement outcomes on the two qubits holds true no matter which rotated basis a rotated basis $|v\rangle, |v^\perp\rangle$ the two qubits are measured in, where $|0\rangle = \alpha|v\rangle + \beta|v^\perp\rangle$ and $|1\rangle = -\beta|v\rangle + \alpha|v^\perp\rangle$. Thus can be seen as,

$$\begin{aligned} |\phi^+\rangle &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \\ &= \frac{1}{\sqrt{2}}((\alpha|v\rangle + \beta|v^\perp\rangle) \otimes (\alpha|v\rangle + \beta|v^\perp\rangle)) \\ &\quad - \frac{1}{\sqrt{2}}((- \beta|v\rangle + \alpha|v^\perp\rangle) \otimes (- \beta|v\rangle + \alpha|v^\perp\rangle)) \\ &= \frac{1}{\sqrt{2}}((\alpha^2 + \beta^2)|vv\rangle + (\alpha^2 + \beta^2)|v^\perp v^\perp\rangle) \\ &= \frac{1}{\sqrt{2}}((\alpha^2 + \beta^2)|vv\rangle + (\alpha^2 + \beta^2)|v^\perp v^\perp\rangle) \\ &= \frac{1}{\sqrt{2}}(|vv\rangle + |v^\perp v^\perp\rangle) \end{aligned}$$

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. Then 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 $|1\rangle$. Thus, 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, upon measurement we will 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. Thus, they are not entangled.

6.2 Bell States/EPR Pairs (EPR Paradox)

Everyone has heard Einstein's famous quote "God does not play dice with the Universe". The quote is a summary of the following passage from Einstein's 1926 letter to Max Born: "Quantum mechanics is certainly imposing. But an inner voice tells me that it is not yet the real thing. The theory says a lot, but does not really bring us any closer to the secret of the Old One. I, at any rate, am convinced that He does not throw dice." Even to the end of his life, Einstein held on to the view that quantum physics is an incomplete theory and that some day we would learn a more complete and satisfactory theory that describes nature.

In what sense did Einstein consider quantum mechanics to be incomplete? Think about flipping a coin. For all common purposes, the outcome of a coin toss is random — heads half the time, and tails the other half. And this lines up exactly with our observations, but we know that randomness isn't the whole story. A more complete theory would say that if we knew all of the initial conditions of the coin exactly (position, momentum), then we could use Newton's laws of classical physics to figure out exactly how the coin would land, and therefore the outcome of the coin flip. Another way to say this is that the coin flip amplifies our lack of knowledge about the state of the system, and makes the outcome seem completely random. In the same way, Einstein believed that the randomness of quantum measurements reflected our lack of knowledge about additional degrees of freedom, or "hidden variables," of the quantum system.

Einstein sharpened this line of reasoning in a paper he wrote with Podolsky and Rosen in 1935, where they introduced the famous Bell states. The EPR argument works like this. For Bell state, $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, when you measure first qubit (in the bit basis), the second qubit is determined (in the bit basis). What is even more remarkable is that if you measure the first qubit in the sign basis, the second qubit is determined in the sign basis. You should verify that in the sign basis ($|+\rangle, |-\rangle$), the state $|\Phi^+\rangle$ can be written as $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|++\rangle + |--\rangle)$. It follows that knowledge of the sign of one qubit completely determines the other.

Now let's suppose the qubits are very far apart, say one light-second. If we measure qubit 1 in the standard basis, then measure qubit 2 a half second later in the same basis, the two measurements must agree. Then qubit 2 must have been in a definite state for a half second before it was measured: from the instant we measured qubit 1, we knew qubit 2. But the qubits couldn't have communicated any information in that time.

What if we had measured the first qubit in the $|+\rangle, |-\rangle$ basis instead? Then similarly for half a second before we measure qubit 2, it was in a definite state in the $|+\rangle, |-\rangle$ basis. But qubit 2 could not possibly know which basis qubit 1 was measured in until a full second after we measure qubit 1! This is because we assumed that light takes a one second to travel from qubit 1 to qubit 2. This appears to contradict the uncertainty principle for $|\pm\rangle$ and $|0\rangle, |1\rangle$ states says that there is no definite $|\pm\rangle$ state that is also a definite $|0\rangle, |1\rangle$ state.

Einstein, Podolsky, and Rosen concluded that since qubit 2 cannot have any information about which basis qubit 1 was measured in, its state in both bit and sign bases is simultaneously determined, something that quantum mechanics does not allow. EPR therefore suggested that quantum mechanics is an incomplete theory, and there is a more complete theory where "God does not throw dice." Until his death in 1955, Einstein tried to formulate a more complete "local hidden variable theory" that would describe the predictions of quantum mechanics, but without resorting to probabilistic outcomes.

But in 1964, almost three decades after the EPR paper, John Bell showed that properties of Bell (EPR) states were not merely fodder for a philosophical discussion, but had verifiable consequences: local hidden variables are not the answer. He described an experiment to be performed on two qubits entangled in a Bell state such that a local hidden variable theory would disagree with quantum mechanics about the outcome. The Bell experiment has been performed to increasing accuracy, originally by Aspect, and the results have always been consistent with the predictions of quantum mechanics and inconsistent with local hidden variable theories.

Example 6.2.1. Prove that $|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \neq |a\rangle |b\rangle$ for all single qubit state $|a\rangle$ and $|b\rangle$.

Let us assume that the bell state $\frac{|00\rangle + |11\rangle}{\sqrt{2}}$ can be expressed as $|a\rangle \otimes |b\rangle$. Thus,

$$|a\rangle \otimes |b\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

Let us further simplify $|a\rangle = \alpha_a |0\rangle + \beta_a |1\rangle$, and $|b\rangle = \alpha_b |0\rangle + \beta_b |1\rangle$. Upon substi-

tuting, we get,

$$(\alpha_a |0\rangle + \beta_a |1\rangle)(\alpha_b |0\rangle + \beta_b |1\rangle) = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

$$\alpha_a\alpha_b |00\rangle + \alpha_a\beta_b |01\rangle + \beta_a\alpha_b |10\rangle + \beta_a\beta_b |11\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

Thus, comparing equations we get,

$$\begin{aligned}\alpha_a\alpha_b &= \frac{1}{\sqrt{2}} \\ \alpha_a\beta_b &= 0 \\ \beta_a\alpha_b &= 0 \\ \beta_a\beta_b &= \frac{1}{\sqrt{2}}\end{aligned}$$

Thus, two of the equations in the middle requires one of coefficients to be zero but the first and the last equation does not allow any of the coefficients to be zero which is a contradiction. Thus, there does not exist any such set of complex numbers which satisfies all the four equations. Thus, it cannot be factorized. Thus, the Bell state $\frac{|00\rangle + |11\rangle}{\sqrt{2}}$ cannot be expressed as $|a\rangle \otimes |b\rangle$, where $|a\rangle$ and $|b\rangle$ are single-qubit states.

6.3 Bell's Thought Experiment

Bell considered the following experiment: Let us assume that two particles are produced in the bell state $|\Phi^+\rangle$ in a laboratory, and they fly in opposite directions to two distant laboratories. Upon arrival, each of the two qubits is subject to one of two measurements. The decision about which if the two measurements to be performed at each lab is made randomly at the last moment, so that speed o light consideration rule out information about the choice at one lab being transmitted to the other. The measurements are cleverly chosen to distinguish between the predictions of quantum mechanics and any local hidden variable theory. Concretely, the experiment measures the correlation between the outcomes of two experiments. The choice of measurements is such that any classical hidden variable theory predicts that the correlation between the two outcomes can be at most 0.75, whereas quantum mechanics predicts that the correlation is $\cos^2 \pi/8 \approx 0.85$. Thus, the experiment allows us to distinguish between the predictions of quantum mechanics and any local hidden variable theory. We now describe the experiment in more detail.

The two experimenters Alice and Bob each receives one qubit of a Bell state $|\Phi^+\rangle$, and measures it in one of the two bases depending upon the random bit r_A and r_B respectively. Denote by a and b respectively the outcomes of the measurements. We are interested in the highest achievable correlation between the two quantities $r_A \times r_B$ and $a + b(\text{mod}2)$. We will see below that there is a particular choice of bases for the quantum measurements made by A and B such that $P[r_A \times r_B = a + b(\text{mod}2)] = \cos^2 \pi/8 \approx 0.85$. Before we do so, let us see why no classical hidden variable theory allows a correlation of over 0.75. i.e. $P[r_A \times r_B = a + b(\text{mod}2)] \leq 0.75$.

We can no longer postpone a discussion about what a local hidden variable theory is. Let us do so in the context of the Bell experiment. In a local hidden variable theory, when the Bell state was created, the two particles might share an arbitrary amount of classical information x . This information could help them coordinate their responses to any measurements they are subjected to in the future. By design, the Bell experiment selects the random bits r_A and r_B only after two particles are too far apart to exchange any further information before they are measured. Thus we are in the setting, where A and B share some arbitrary classical information x , and are given as input independent, random bits x_A and x_B as input, we must output bits a and b respectively to maximize their chance of achieving $r_A \times r_B = a + b(\text{mod}2)$. It can be shown that the shared information x is of no use in increasing this correlation, and indeed, the best they can do is to always output $a = b = 0$. This gives $Pr[r_A \times r_B = a_b(\text{mod}2)] \leq 0.75$.

Let us now describe the quantum measurements that achieve greater correlation. They are remarkably simple to describe:

- if $r_A = 0$, then Alice measures in the standard $|0\rangle / |1\rangle$ basis.
- if $r_A = 1$, then Alice measures in the $\pi/4$ basis (i.e. standard basis rotated by $\pi/4$)
- if $r_B = 0$, then Bob measures in the $\pi/8$ basis.
- if $r_B = 1$, the Bob measures in the $-\pi/8$ basis.

The analysis of the success probability of this experiment is also beautifully simple. We will show that in each of the four cases $r_A = r_B = 0$, etc, the success probability $P[r_A \times r_B = a + b(\text{mod}2)] = \cos^2 \pi/8$.

We first note that if Alice and Bob measure in bases that make an angle θ with each other, then the chance that their measurement outcomes are the same (bit) is exactly $\cos^2 \theta$. This follows from the rotational invariance of $|\Phi^+\rangle$ and the following observation: if the first qubit is measured in the standard basis, then the outcome

is an unbiased bit. Moreover the state of the second qubit is exactly equal to the outcome of the measurement - $|0\rangle$ if the measurement outcome is 0, say. But now if the second qubit is measured in a basis rotated by θ , then the probability that the outcome is also 0 is exactly $\cos^2 \theta$.

Now observe that in three of the four cases, where $x_A \cdot x_B = 0$. Alice and Bob measure in bases that make an angle of $\pi/8$ with each other. By our observation above, $P[a + b \equiv 0 \pmod{2}] = P[a = b] = \cos^2 \pi/8$.

In the last case $x_A \cdot x_B = 1$, and they measure in bases that make an angle of $3\pi/8$ with each other. So, $P[a + b \equiv 0 \pmod{2}] = P[a \neq b] = \cos^2 3\pi/8 = \sin^2(\pi/2 - 3\pi/8) = \sin^2 \pi/8$. Therefore, $P[a = b \equiv 1 \pmod{2}] = 1 - \sin^2 \pi/8 = \cos^2 \pi/8$. So in each of the four cases, the chance that Alice and Bob succeed is $\cos^2 \pi/8 = 0.85$.

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 6.1. 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 $\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |1\rangle = \frac{|01\rangle + |11\rangle}{\sqrt{2}}$, and the state after applying the operation CX will be $\frac{|01\rangle + |10\rangle}{\sqrt{2}} = \beta_{01}$. Similarly, we can create all the other Bell states. **Note that the Bell states $|\Phi^\pm\rangle = \frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle)$; $|\Psi^\pm\rangle = \frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle)$ form an orthonormal basis for \mathbb{C}^4 .**

Example 6.3.1. 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

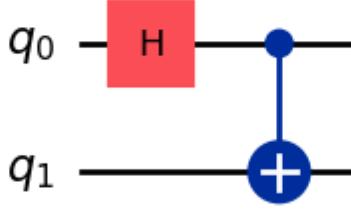


Figure 6.1: Circuit for creating Bell States

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. 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

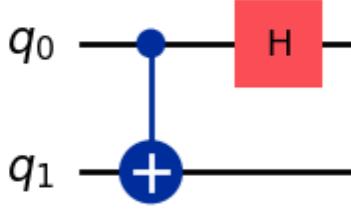


Figure 6.2: Circuit for Reversing Bell States

first qubit and the circuit will give back the original input state. (Hadamard gate is also self-inverse).

Example 6.3.2. Verify that the Bell basis forms an orthonormal basis for the two qubit state space.

We know that

$$|\beta_{xy}\rangle = \frac{|0y\rangle + (-1)^x |1\bar{y}\rangle}{\sqrt{2}}$$

where $x, y \in \{0, 1\}$. Taking inner product with another bell state, we get,

$$\begin{aligned} \langle \alpha_{pq} | \beta_{xy} \rangle &= \left(\frac{\langle 0q | + (-1)^p \langle 1\bar{q} |}{\sqrt{2}} \right) \left(\frac{|0y\rangle + (-1)^x |1\bar{y}\rangle}{\sqrt{2}} \right) \\ &= \frac{\langle 0q | 0y \rangle + (-1)^x \langle 0q | 1\bar{y} \rangle + (-1)^p \langle 1\bar{q} | 0y \rangle + (-1)^{x+p} \langle 1\bar{q} | 1\bar{y} \rangle}{2} \end{aligned}$$

Let $p = x$ and $q = y$, the above equation simplifies to, using the fact that $\langle 0q | 0y \rangle = \langle 1\bar{q} | q\bar{y} \rangle = 1$, else, $\langle 0q | q\bar{y} \rangle = \langle 1\bar{q} | 0y \rangle = 0$, and using the fact that $x = p \implies x + p = \text{even}$, we get,

$$\begin{aligned} &= \frac{1 + 0 + 0 + (-1)^{x+p} 1}{2} \\ &= 1 \end{aligned}$$

Consider either $x \neq p$ or $y \neq q$ or both. In all the possible combinations of these cases, we would get, the value as 0. Hence, the bell states form an orthonormal basis for two qubit states.

Recall, that we can write any vector $|\psi\rangle$ in the 2-qubit state space as:

$$|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$$

as:

$$\left(\frac{\alpha+\delta}{\sqrt{2}}\right)|\beta_{00}\rangle + \left(\frac{\alpha-\delta}{\sqrt{2}}\right)|\beta_{01}\rangle + \left(\frac{\beta+\gamma}{\sqrt{2}}\right)|\beta_{10}\rangle + \left(\frac{\beta-\gamma}{\sqrt{2}}\right)|\beta_{11}\rangle$$

hence, this shows that the Bell states form a basis.

6.4 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 6.4.1. *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

$$\langle\phi|0|\psi\rangle = \alpha_0^*\beta_0 + \alpha_1^*\beta_1$$

Their inner product after the transformation will be:

$$\langle\phi\phi|\psi\psi\rangle = (\alpha_0^*\beta_0 + \alpha_1^*\beta_1)^2 = \langle\phi|0|\psi\rangle^2 = \langle\phi|\psi\rangle^2$$

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

Example 6.4.1. Explain how a device which, upon input of one of two non-orthogonal states $|\psi\rangle$ or $|\varphi\rangle$ correctly identified the state, could be used to build a device which cloned the states $|\psi\rangle$ and $|\varphi\rangle$, in violation of the no-cloning theorem. Conversely, explain how a device which cloned the states $|\psi\rangle$ and $|\varphi\rangle$, in violation of the no-cloning theorem. Conversely, explain how a device for cloning could be used to distinguish non-orthogonal quantum states.

Given access to a device which can distinguish non-orthogonal states $|\psi\rangle, |\varphi\rangle$ (without measurement), we can then design a quantum circuit that would map $|\psi\rangle \rightarrow |\varphi\rangle$ (or vice versa), allowing us to clone the states we like.

Conversely, given a cloning device, we could clone $|\psi\rangle$ and $|\varphi\rangle$ an arbitrary number of times. Then, performing repeated measurements of the two states based on the measurement statistics in different measurement bases, we would (given enough measurements) be able to distinguish the two states based on the measurement statistics (there will of course be some error ϵ based on probabilistic considerations, but given that we have access to as many measurements of the states as we like, we are able to make this error arbitrarily low).

6.5 Applications of Entanglement

6.5.1 The CHSH Game

When we speak of a game in this context, we are not talking about something that's meant to be played for fun or sport, but rather a mathematical abstraction in the sense of game theory. Mathematical abstractions of games are studied in economics and computer science, for instance, and have great utility.

The letters CHSH refer to the authors - John Clauser, Micheal Horne, Abner Shimony, and Richard Holt - of a 1969 paper where the example was first described. They did not describe the example as a game, but rather as an experiment. Its description as a game, however, is both natural and intuitive.

The CHSH game falls within a class of games known as non local games. Nonlocal games are fascinating and have deep connections to physics, computer science, and mathematics - holding mysteries that still remain unsolved. We'll begin the section by explaining what nonlocal games are, and then we'll focus in on the CHSH game and what makes it interesting.

Nonlocal games

A nonlocal game is a cooperative game where two players, Alice and Bob, work together to achieve a particular outcome. The game is run by a referee, who behaves according to strict guidelines that are known to Alice and Bob.

Alice and Bob can prepare for the game however they choose, but once the game starts they are forbidden from communicating. We might imagine the game taking place in a secure facility of some sort - as if the referee is placing the role of a detective and Alice and Bob are suspects being interrogated in different rooms. but another way of thinking about the set-up is that Alice and Bob are separated by a vast distance, and communication is prohibited because the speed of light doesn't allow for it within the running time of the game. That is to say, if Alice tries to send a message to Bob, the game will be over by the time he receives it, and vice versa.

The way a nonlocal game works is that the referee first asks each of Alice and Bob a question. We'll use the letter x to refer to Alice's questions and y to refer to Bob's question. Here we're thinking of x and y as being classical states, and in the CHSH game x and y are bits.

The referee uses randomness to select these questions. To be precise there is some probability $p(x, y)$ associated with each possible pair (x, y) of questions, and the referee has vowed to choose the questions, at the time of the game, in this way. Everyone including Alice and Bob, knows these probabilities - but nobody knows specifically which pair (x, y) will be chosen until the game begins.

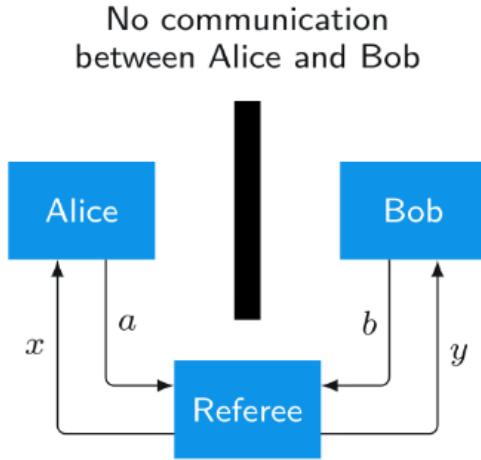
After Alice and Bob receive their questions, they must then provide answers: Alice's answer is a and Bob's answer is b . Again, these are classical states in general, and bits in the CHSH game.

At this point the referee makes a decision: Alice and Bob either win or loose depending on whether or not the pair of answers (a, b) is deemed correct for the pair of questions (x, y) according to some fixed set of rules. Different rules mean different games, and the rules for the CHSH game specifically are described in the section following this one. As was already suggested, the rules are known to everyone.

The following diagram provides a graphic representation of the interactions.

it is the uncertainty about which questions will be asked, and specifically the fact that each player doesn't know the other player's question, that makes non local games challenging for Alice and Bob - just like colluding suspects in different rooms trying to keep their story straight.

A precise description of the referee defines an instance of a nonlocal game. This includes a specification of the probabilities $p(x, y)$ for each question pair along with the rules that determine whether a pair of answers (a, b) wins or loses for each possible question pair (x, y) .



We'll take a look at the CHSH game momentarily, but it's also interesting to consider other non local games. Some non local games are fascinating and mysterious to this day, and some aren't so interesting. While the set-up is simple, there's complexity at work - and for some nonlocal games it can be impossibly difficult to compute best or near best strategies for Alice and Bob. This is part of the fascinating nature of the non-local games model.

CHSH game description

Here is the precise description of the CHSH game, where (as above) x is Alice's question, y is Bob's question, a is Alice's answer, and b is Bob's answer:

1. The questions and answers are all bits: $x, y, a, b \in \{0, 1\}$.
2. The referee chooses the questions (x, y) uniformly at random. That is, each of the four possibilities. $(0, 0), (0, 1), (1, 0), (1, 1)$ is selected with probability $1/4$.
3. The answers (a, b) win for the questions (x, y) if $a \oplus b = x \wedge y$ and lose otherwise. The following table expresses this rule by listing the winning and losing conditions on the answers (a, b) for each pair of questions (x, y) .

Limitations of classical strategies

Now let's consider strategies for Alice and Bob in the CHSH game, beginning with classical strategies.

(x, y)	win	lose
$(0, 0)$	$a = b$	$a \neq b$
$(0, 1)$	$a = b$	$a \neq b$
$(1, 0)$	$a = b$	$a \neq b$
$(1, 1)$	$a \neq b$	$a = b$

Deterministic Strategies

We'll start with deterministic strategies, where Alice's answer a is a function of the question x that she receives, and likewise Bob's answer b is a function of the question y he receives. So, for instance, we may write $a(0)$ to represent Alice's answer when her question is 0, and $a(1)$ to represent Alice's answer when her question is 1.

No deterministic strategy can possibly win the CHSH game every time. One way to respond to this is simply to go one-by-one through all of the possible deterministic strategies and check that every one of them losses for at least one of the four possible question pairs. Alice and Bob can each choose from four possible question pairs. Alice and Bob can each choose from four possible functions from one bit to one bit - which we encountered and so there are 16 different deterministic strategies in total to check.

We can also reason this fact analytically. If Alice and Bob's strategy wins when $(x, y) = (0, 0)$, then it must be that $a(0) = b(0)$; if their strategy wins for $(x, y) = (0, 1)$, we conclude that $a(0) = b(0)$; if their strategy wins for $(x, y) = (0, 1)$, we conclude that $a(0) = b(1)$; and similarly, if the strategy wins for $(x, y) = (1, 0)$, then $a(1) = b(0)$. So, if their strategy wins for all three possibilities, then we have

$$b(1) = a(0) = b(0) = a(1)$$

This implies that the strategy loses in the final case $(x, y) = (1, 1)$, for here winning requires that $a(1) \neq b(1)$.

Thus, there can be no deterministic strategy that wins every time. It's easy to find strategies that win in three of the four cases (such as $a(0) = a(1) = b(0) = b(1) = 0$), and so we see that the maximum probabilities for Alice and Bob to win using a deterministic strategy is 3/4.

Probabilistic Strategies

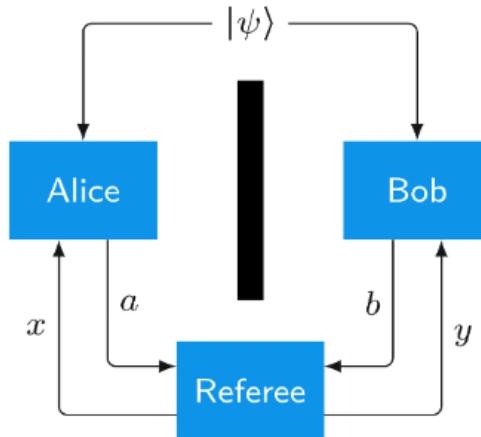
As we just concluded, Alice and Bob cannot do better than winning the CHSH game 75% of the time using a deterministic strategy. But what about a probabilistic strategy? Could it help Alice and Bob to use randomness - including the possibility of shared randomness, where their random choices are correlated?

It turns out that probabilistic strategies don't help at all to increase the probability that Alice and Bob win. This is because every probabilistic strategy can alternatively be viewed as a random selection of a deterministic strategy, just like probabilistic operations can be viewed as random selections of deterministic operations. The average is never larger than the maximum, and so it follows that probabilistic strategies don't offer any advantage in terms of their overall winning probability.

Thus, winning with probability $3/4$ is the best that Alice and Bob can do using any classical strategy, whether deterministic or probabilistic.

CHSH game strategy

A natural question to ask at this point is whether Alice and Bob can do any better using a quantum strategy. In particular, if they share an entangled quantum state as the following figure suggests, can they increase their winning probability? The



answer is yes, and this is the main point of the example and why it's so interesting. So let's see exactly how Alice and Bob can do better in this game using entanglement.

Required vectors and matrices

The first thing we need to do is define a qubit state vector $|\psi - \theta\rangle$, for each real number θ (which we'll think of as an angle measured in radians) as follows.

$$|\psi_\theta\rangle = \cos(\theta)|0\rangle + \sin(\theta)|1\rangle$$

Here are some simple examples:

$$\begin{aligned} |\psi_0\rangle &= |0\rangle \\ |\psi_{\pi/2}\rangle &= |1\rangle \\ |\psi_{\pi/4}\rangle &= |+\rangle \\ |\psi_{-\pi/4}\rangle &= |-\rangle \end{aligned}$$

We also have the following examples, which arise in the analysis below:

$$\begin{aligned} |\psi_{-\pi/8}\rangle &= \frac{\sqrt{2+\sqrt{2}}}{2}|0\rangle - \frac{\sqrt{2-\sqrt{2}}}{2}|1\rangle \\ |\psi_{\pi/8}\rangle &= \frac{\sqrt{2+\sqrt{2}}}{2}|0\rangle + \frac{\sqrt{2-\sqrt{2}}}{2}|1\rangle \\ |\psi_{-3\pi/8}\rangle &= \frac{\sqrt{2-\sqrt{2}}}{2}|0\rangle + \frac{\sqrt{2+\sqrt{2}}}{2}|1\rangle \\ |\psi_{5\pi/8}\rangle &= -\frac{\sqrt{2-\sqrt{2}}}{2}|0\rangle + \frac{\sqrt{2+\sqrt{2}}}{2}|1\rangle \end{aligned}$$

Looking at the general form, we see that the inner product between any two of these vectors has this formula:

$$\langle\psi_\alpha|\psi_\beta\rangle = \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta) = \cos(\alpha - \beta)$$

In detail, we only have real number entries in these vectors, so there are no complex conjugates to worry about: the inner product is the product of the cosines plus the product of the sines. We can then use one of the so-called angle addition formulas to simplify. This formula reveals the geometric interpretation of the inner product between real unit vectors, as the cosine of the angle between them.

Now, if we compute the inner product of the tensor product of any two of these vectors with the $|\phi^+\rangle$ state we obtain a similar expression, except that it has a $\sqrt{2}$ in the denominator.

Implementation using Qiskit

6.5.2 Quantum Teleportation

Suppose Alice has a qubit that she wants to send to Bob. Let us say that the state of the qubit is $\alpha|0\rangle + \beta|1\rangle$. How many classical bits would be required to accomplish this task?

There are two reasonable answers to the question. the first answer is that the number of classical bits depends on the desired precision: α and β are arbitrary complex numbers (with $|\alpha|^2 + |\beta|^2 = 1$), but Alice could send approximations of α and β to Bob. The amplitudes α and β cold require an infinite number of bits of precision to write them down exactly.

The second answer is that no number of classical bits of communication will suffice. Alice may not know α and β , and there is no way for her to perform measurements on her qubit that would reveal these numbers. Thus, she could not communicate this information to Bob because she could not even determine it for herself. There is also the possibility that Alice's qubit is entangled with one or more qubits (in which case of course we would not be able to describe the state of the qubit as $\alpha|0\rangle = \beta|1\rangle$). Classical communication from Alice to bob would not be able to somehow transmit this entanglement.

However, if we give Alice and Bob the additional resource of sharing a pair of qubits from bell states, then it becomes possible for Alice to transmit a qubit to Bob using classical communication by means of quantum teleportation. Specifically, two bits of classical information will be needed to perform this task (two classical bits need to be told to Bob). Depending upon which of four outcomes Alice announces to him, Bob performs one of four operations on his qubit, and voila, his qubit is in the required state.

Note that this doesn't violate the no cloning theorem because Alice's qubit was destroyed by measurement protocol before Bob created his copy.

They get together and create a Bell state β_{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. 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 β_{00} . It is as shown in the figure 6.3 which has both the inputs $q_0 = |0\rangle$ and $q_1 = |0\rangle$. 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 6.4 with $q_0 = |\psi\rangle$ and $q_1 = |\beta_A\rangle$.

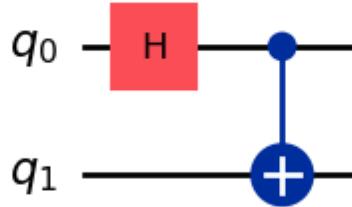


Figure 6.3: Creating Bell States

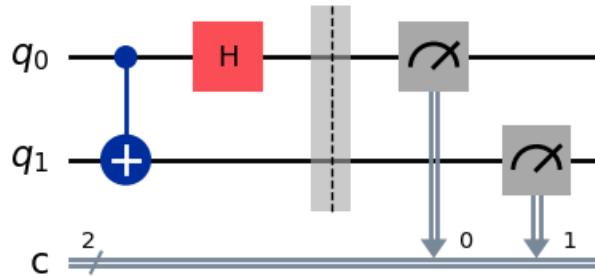


Figure 6.4: Alice's Operations

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 $CNOT \otimes I$ with first bit as control bit and the second bit as the target bit and no operation on the third bit ($|\beta_B\rangle$ which is with Bob), 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, $(H \otimes I \otimes I)$, 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 factorizing 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. Recall 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 $\frac{1}{4}$.

Note that there have been no measurements or communication at this point. Bob's qubit seems to depend on α and β , but this is not really so. There would be no way for Bob to transform and measure his qubit alone at this point so that he would learn anything about α and β . The state above has been expressed in a convenient way to determine the distribution of measurement outcomes and the resulting state of Bob's qubit after the measurements. Suppose upon measuring in the computational basis 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.

$$(I \otimes I \otimes Z)(\alpha|100\rangle - \beta|101\rangle) = \alpha|100\rangle + \beta|101\rangle = |10\rangle \otimes (\alpha|0\rangle + \beta|1\rangle)$$

- 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.

$$(I \otimes I \otimes X)(\alpha|011\rangle + \beta|010\rangle) = \alpha|010\rangle + \beta|011\rangle = |01\rangle \otimes (\alpha|0\rangle + \beta|1\rangle)$$

- 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.

$$(I \otimes I \otimes ZX)(\alpha|111\rangle - \beta|110\rangle) = \alpha|110\rangle + \beta|111\rangle = |11\rangle \otimes (\alpha|0\rangle + \beta|1\rangle)$$

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. Note that in the figure 6.5 circuit diagram we see that the measurement results in a classical bits and the gates then act on one classical bit and one qubit. generally we will only do this when the classical bit is a control bit for some operation. all that this means is that if the (classical) control bit is 1 then perform the corresponding operation on the qubit (σ_Z or σ_X), otherwise do nothing.

To summarise, we start with the following scenario. Alice and Bob share two qubits in the state $a|00\rangle + b|11\rangle$. Alice and Bob don't know the amplitudes a and b . how can Bob end up with the state $a|0\rangle + b|1\rangle$? An easy way to achieve this is to perform a CNOT gate on the two qubits with Bob's qubit as the control, and Alice's qubit as the target. But this requires an exchange of quantum information. What if Alice and Bob can only exchange classical information?

The way is Alice performs a hadamard on her qubit. The state of the two qubits is now $a/\sqrt{2}(|0\rangle + |1\rangle)|0\rangle + b/\sqrt{2}(|0\rangle - |1\rangle)|1\rangle = 1/\sqrt{2}|0\rangle(a|0\rangle + b|1\rangle) + 1/\sqrt{2}|1\rangle(a|1\rangle - b|0\rangle)$

$b|1\rangle$). Now if Alice measures her qubit in the standard basis, if the measurement outcome is 0, the Bob's qubit is desired $a|0\rangle + b|1\rangle$. If the measurement outcomes is 1, then Bob's qubit is $a|0\rangle - b|1\rangle$. But in this case if Bob were to apply a phase flip gate (Z) to his qubit, it would end up in the desired state $a|0\rangle + b|1\rangle$,

Back to Teleportation. Alice has a qubit in the state $a|0\rangle = b|1\rangle$, and Alice and Bob share a Bell state. Is there a way for them to convert their joint state to $a|00\rangle + b|11\rangle$, without exchanging any quantum information? If they succeed, then by our previous discussion Alice can teleport her qubit to Bob. Consider what happens if Alice applies a CNOT gate with her qubit $a|0\rangle + b|1\rangle$ as the control qubit, and her share of the Bell state as the target qubit.

$$|\phi\rangle \otimes |\psi\rangle = \sum_{i=0,1} a_i |i\rangle \otimes \sum_{j=0,1} \frac{1}{\sqrt{2}} |j,j\rangle$$

After passing through the CNOT gate this becomes

$$\sum_{i,j} a_i |i, i \oplus l\rangle$$

Now A measures the middle qubit. Suppose it is measured as l ; then $l = i \oplus j \implies j = i \oplus l$. The state is now

$$\sum_i a_i |i, i \oplus l\rangle$$

next, A transmits L to B . If $l = 0$, B takes no action, while if $l = 1$, then B performs a bit flip X , on his qubit, resulting in the desired state

$$\sum_i a_i |i, i\rangle$$

We already saw how to teleport once we achieved this state. Putting it all together, the following quantum circuit 6.5 circuit describes the resulting teleportation protocol. The topmost qubit is the unknown qubit that Alice wishes to teleport, the second and third qubits are initially in a Bell state. The measurement of the middle qubit after performing the CNOT gate sets up the state $\sum_j a_j |j, j\rangle$ on the first and third qubit. Moreover, A' application of the Hadamard gate on the first qubit induces the transformation

$$\sum_j a_j |j, j\rangle \rightarrow \sum_{ij} a_j (-1)^{ij} |i, j\rangle$$

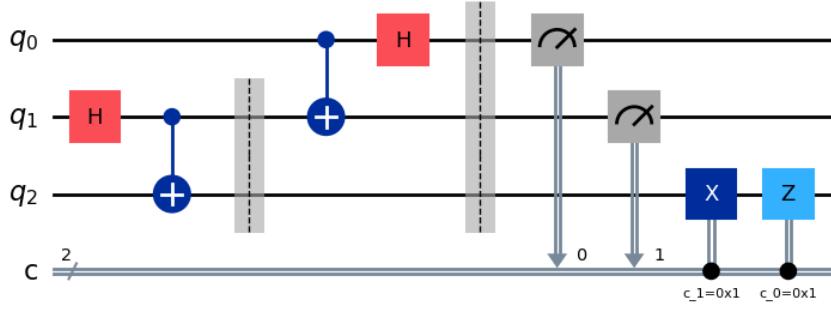


Figure 6.5: Teleportation Circuit

Finally A measures i and sends the measurement to B. The state is now:

$$\sum_j a_j (-1)^{ij} |j\rangle$$

If $i = 0$ then we are done; if $i = 1$ then B applies a phase flip. In either case the state is now $a_0 |0\rangle + a_1 |1\rangle$. So A has transported the quantum state to B simply by sending two classical bits. The final Teleportation circuit is as shown in the figure 6.5. 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 6.5, show that after Alice's measurement the bits act as control bits for the operations to be applied by Bob.

Something stronger is actually true. If Alice's initial qubit was entangled with other qubits, this entanglement will be preserved. In other words, teleportation works like a perfect quantum channel - it is exactly as if Alice had physically sent her qubit to Bob. **If Alice's qubit had been entangled with some other qubits, then teleportation preserves this entanglement: Bob then receives a qubit that is entangled in the same way as Alice's original qubit was.** Note that qubit on Alice's side has been destroyed: teleporting moves a qubit from Alice to Bob, rather than copying it.

Implementation using Qiskit

Background

Usage estimate: 4 seconds on ibm_kyoto (Note: This is an estimate. Your runtime may vary).

In order to demonstrate

6.5.3 Superdense Coding

Suppose Alice and Bob are connected by a quantum communications channel. By this we mean, for example, that they cannot communicate qubits over an optical fibre using polarized photons. Is this, much more powerful than a classical communication channel, over which only classical bits may be transmitted? The answer seems obvious, since a classical bit is a special case of quantum bit. And a qubit appears to encode an infinite number of bits of information, since to specify this state we must specify two complex numbers. However, the truth is a little more subtle, since the axioms of quantum mechanics also restrict how we may access information about the quantum state by measurement. So the question is “how many classical bits can Alice transmit to Bob in a message consisting of a single qubit?”. We will show that if Alice and Bob share entanglement in the form of a Bell state, then Alice can transmit two classical bits by transmitting just one qubit over the quantum channel.

The overall idea is this: say Alice and Bob share $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. Alice can transform this shared state to any of the four Bell basis state $|\Phi^+\rangle, |\Phi^-\rangle, |\Psi^+\rangle, |\Psi^-\rangle$ by applying a suitable quantum gate just to her qubit. now if she transmits her qubit to bob, he holds both qubit of a bell basis state and can perform a measurement in the Bell basis to distinguish which of the four states he holds.

Let’s now see the details of Alice’s protocol: if Alice wishes to transmit the two bit messages $b_1 b_2$, she applies a bit flip X to her qubit if $b_1 = 1$ and a phase flip Z to her qubit if $b_2 = 1$. you should verify that in the four cases 00, 01, 10, 11 this results in the two being in the state $|\Phi^+\rangle, |\Phi^-\rangle, |\Psi^+\rangle, |\Psi^-\rangle$ respectively.

After receiving Alice’s qubit, Bob measures the two qubits in the bell basis by running the circuit shown in figure 6.8 ($(H \otimes I) \cdot CNOT$), then measuring in the standard basis.

Note that Alice did really use the two qubits total to transmit the two classical bits. After all, Alice and Bob somehow had to start with a shared Bell State. However, the first qubit - Bob’s half of the bell state - could have been sent well before Alice had decided what message she wished to send to Bob. It is natural to view entanglement as a resource as we will see; and when Alice and Bob each have a qubit and the two of them are in the state above, it is natural to imagine that Alice

and Bob share one unit (i.e.e one entangled bit, or e-bit for short) of entanglement. Given the additional resource of a shared e-bit of entanglement, Alice will be able to transmit both classical bits to Bob by sending just one qubit.

One can show that it is not possible to do any better. No more than two classical bits can be transmitted by sending just one qubit.

In Super dense coding we can send two classical bits from one place to another but using only one qubit. Alice and Bob create a bell state β_{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 β_{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. This is how the Bell

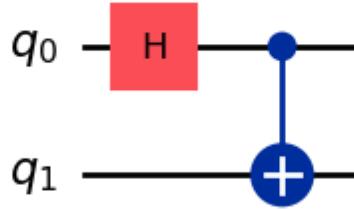


Figure 6.6: Creating Bell States

state is created shown in the figure 6.6.

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 6.7.

- If Alice wishes to send 00, then she applies no operation.

$$(I \otimes I) \frac{|00\rangle + |11\rangle}{\sqrt{2}} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

I : 00
X : 01
Z : 10
ZX: 11

Figure 6.7: Alice's Encoding

- If Alice wishes to send 01, then she applies the Pauli - X gate on her portion of Bell state.

$$(X \otimes I) \left(\frac{|00\rangle + |11\rangle}{\sqrt{2}} \right) = \frac{|10\rangle + |01\rangle}{\sqrt{2}}$$

- If Alice wishes to send 10, then she applies the Pauli-Z gate on the qubit.

$$(Z \otimes I) \left(\frac{|00\rangle + |11\rangle}{\sqrt{2}} \right) = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$

- If Alice wishes to send 11, then she applies the Pauli-X and Pauli-Z gate on the qubit.

$$(XZ \otimes I) \left(\frac{|00\rangle + |11\rangle}{\sqrt{2}} \right) = \frac{|10\rangle - |01\rangle}{\sqrt{2}}$$

In summary, in order to send the two classical bits $b_1 b_2$, Alice applies Z first if $b_1 = 1$ (if $b_1 = 0$ he does not apply Z, he applies I i.e. does nothing) and then applies X if $b_2 = 1$ (if $b_2 = 0$ he does not apply X, he applies I i.e. does nothing).

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 (his 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 circuit for this is as shown in the figure 6.8.

- 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,

$$CNOT \left(\frac{|00\rangle + |11\rangle}{\sqrt{2}} \right) = \frac{|00\rangle + |10\rangle}{\sqrt{2}}$$

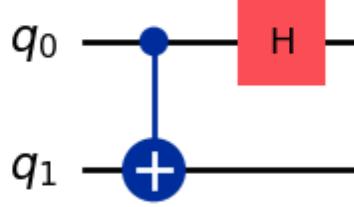


Figure 6.8: Bob's Decoding

Now, after applying the Hadamard Gate on the first qubit we get,

$$(H \otimes I) \left(\frac{|00\rangle + |10\rangle}{\sqrt{2}} \right) = \frac{|00\rangle + |10\rangle + |00\rangle - |10\rangle}{2} = |00\rangle$$

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,

$$(H \otimes I) \left(\frac{|11\rangle + |01\rangle}{\sqrt{2}} \right) = \frac{|01\rangle - |11\rangle + |01\rangle + |11\rangle}{2} = |01\rangle$$

- 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,

$$(H \otimes I) \left(\frac{|00\rangle - |10\rangle}{\sqrt{2}} \right) = \frac{|00\rangle + |10\rangle - |00\rangle + |10\rangle}{2} = |10\rangle$$

- 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,

$$(H \otimes I) \left(\frac{|11\rangle - |01\rangle}{\sqrt{2}} \right) = \frac{|01\rangle - |11\rangle - |01\rangle - |11\rangle}{2} = -|11\rangle$$

Thus, upon measurement the two classical bits are 11.

When Bob measures at the end of the protocol, it is clear that he sees $b_1 b_2$ with certainty.

The final Superdense coding circuit is as shown in the figure 6.9.

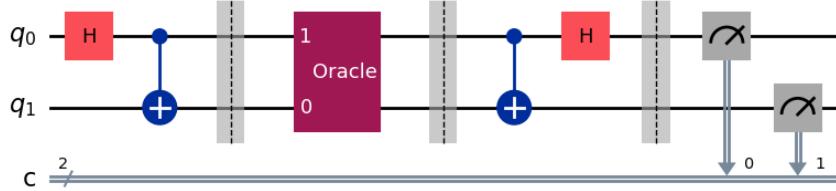


Figure 6.9: Superdense Coding Circuit

Example 6.5.1. Suppose E is any positive operator acting on Alice's qubit. Show that $\langle \psi | E \otimes I | \psi \rangle$ takes the same value when $|\psi\rangle$ is any of the four Bell states. Suppose some maloveloent third party ('Eve') intercepts Alice's qubit on the way to Bob in the superdense coding protocol. Can eve infer anything about which of the four possible bit strings 00, 01, 10, 11 Alice is trying to send? If so how, or if not, why not?

Since E is a positive operator ($\implies E = E^\dagger$ and $\langle x | E | x \rangle \geq 0 \quad \forall x$), it is a hermitian matrix (an obseravable) with all real eigen values greater than 0. Thus, E

is a measurement operator. We can write the above expression as,

$$\begin{aligned}\langle \beta |_{xy} (E \otimes I) |\beta \rangle_{xy} &= \frac{1}{2} (\langle 0y| + (-1)^x \langle 1\bar{y}|) (E \otimes I) (\langle 0y \rangle + (-1)^x \langle 1\bar{y} \rangle) \\ &= \frac{1}{2} (\langle 0| E |0 \rangle \otimes \langle y \rangle + (-1)^x \langle 0| E |1 \rangle \otimes \langle y | \bar{y} \rangle) \\ &\quad + \frac{1}{2} ((-1)^x \langle 1| E |0 \rangle \otimes \langle \bar{y} | y \rangle + (-1)^{2x} \langle 1| E |1 \rangle \otimes \langle \bar{y} | \bar{y} \rangle)\end{aligned}$$

Now since $y \in \{0, 1\}$ and the state $|y\rangle$ is unit norm, we can write $\langle y \rangle = 1$ and $\langle y | \bar{y} \rangle = \langle \bar{y} | y \rangle = 0$ (since $|y\rangle$ and $|\bar{y}\rangle$ will always be orthonormal states). Thus, on substituting this, the above expression simplifies to,

$$\langle \beta |_{xy} (E \otimes I) |\beta \rangle_{xy} = \frac{1}{2} (\langle 0| E |0 \rangle + \langle 1| E |1 \rangle)$$

Thus, irrespective of the values of x and y , the value of $\langle \beta |_{xy} (E \otimes I) |\beta \rangle_{xy} = \langle \psi | (E \otimes I) |\psi \rangle = \frac{1}{2}(\langle 0| E |0 \rangle + \langle 1| E |1 \rangle)$ which is same for all the four bell states (as it does not depend on values of x and y). Thus, for any positive operator (say E) acting on the first qubit of a 2-qubit bell state takes the same value irrespective of the bell state.

$$\langle \beta |_{xy} (E \otimes I) |\beta \rangle_{xy} = \left(\frac{\langle 0| E |0 \rangle + \langle 1| E |1 \rangle}{2} \right)$$

In the superdense coding protocol recall that Alice sends its one qubit (which is in one of the Bell states) to Bob after performing the required operations based on what two classical bits Alice wishes to communicate to Bob. While sending, if an unauthorized third party Eve intercepts Alice's qubit being sent to Bob across the channel then as can be seen from the result in (a), since Eve has only one qubit no matter what observable (hermitian operator) she uses to measure the Alice's qubit being sent, she would always get the same measurement outcome

$$\langle \psi | (E \otimes I) |\psi \rangle = \frac{1}{2}(\langle 0| E |0 \rangle + \langle 1| E |1 \rangle)$$

Since the probability of getting any of the eigen value of E (say m) is same and after the measurement the state of the qubit will be in the Eigen state corresponding to the eigen value m . Thus, no matter what operator Eve uses it will always outcome one of its eigen values with the same probability and the post-measurement state would be the eigen ket corresponding to the outcome (eigenvalue) of the measurement. Thus, Eve cannot infer anything about which of the four possible bit pairs 00, 01, 10, 11 Alice is trying to send.

Implementation using Qiskit

Background

Usage estimate: 4 seconds on ibm_kyoto (NOTE; This is an estimate only. Your runtime may vary)

In order to run the superdense coding circuit as shown in figure 6.9, we are require to program an oracle which will implement the required gates based on the classical bit inputs given by Alice to be transmitted to Bob.

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

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 outputs one of the IBM hardware (say ibm_kyoto).

1. Step 1: Map classical inputs to a quantum problem:

In order to implement an oracle which maps the classical inputs to a quantum problem of superdense coding we use qiskit circuit library to setup a 2-qubit oracle which implements the required gates based on the classical bits as given in figure 6.7. The runtime sampler is used for execution of the circuit.

```

1 def superdense_circuit(string):
2     # Create a Quantum Circuit

```

```

3     qubits=QuantumRegister(2,name="q")
4     cbits=ClassicalRegister(2,name="c")
5     circuit = QuantumCircuit(qubits, cbits)
6
7     q0,q1=qubits
8     c0,c1=cbits
9
10    # Prepare the state to be teleported
11    circuit.h(q0)
12    circuit.cx(q0, q1)
13
14    circuit.barrier()
15
16    if string[0]== "1":
17        circuit.z(q0)
18    if string[1]== "1":
19        circuit.x(q0)
20
21    circuit.barrier()
22
23    circuit.cx(q0, q1)
24    circuit.h(q0)
25
26    circuit.barrier()
27
28    circuit.measure(q0, c0)
29    circuit.measure(q1, c1)
30
31    return circuit
32

```

Note that the above code implements the oracle based on the input given. It can be seen that it clearly implements the oracle according to the Alice's encoding based on the classical bits she wishes to transmit to Bob according to the figure 6.7.

```

1 qc_00=superdense_circuit("00")
2 qc_01=superdense_circuit("01")
3 qc_10=superdense_circuit("10")
4 qc_11=superdense_circuit("11")
5

```

The above code will create quantum circuits which will encode Alice's classical bits to be transmitted to Bob into the circuit using the oracle.

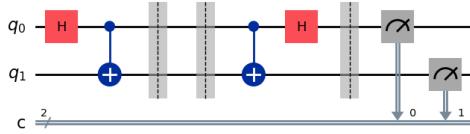
```

1 qc_00.draw('mpl')

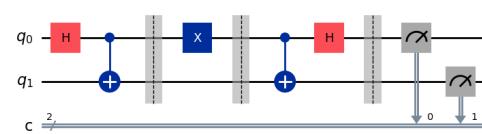
```

2

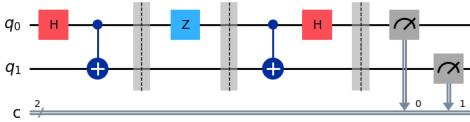
Similarly any other circuit can be created by changing the syntax parameters. The circuit along with the corresponding oracle implementing the function is as shown in the figure 6.10. Note that the oracle (shown between the two



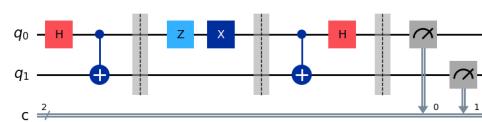
(a) Classical bits: 00



(b) Classical bits: 01



(c) Classical bits: 10



(d) Classical bits: 11

Figure 6.10: Superdense Coding Circuits based on the classical bits Alice wishes to transmit to Bob

barriers) implements the required gates corresponding to the classical bits. Thus the above shown figure corresponds to the Superdense coding circuit.

2. Step 2: Optimize Problems for Quantum Execution

```

1 from qiskit.transpiler.preset_passmanagers import
   generate_preset_manager
2
3 target=backend.target
4 pm = generate_preset_manager(backend=backend,
   optimization_level=3)
5
6 circuit_is_a_00=pm.run(qc_00)
7 circuit_is_a_00.draw(output="mpl", idle_wires=False, style="iqp")

```

The Optimized circuit shown in figure 6.11 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

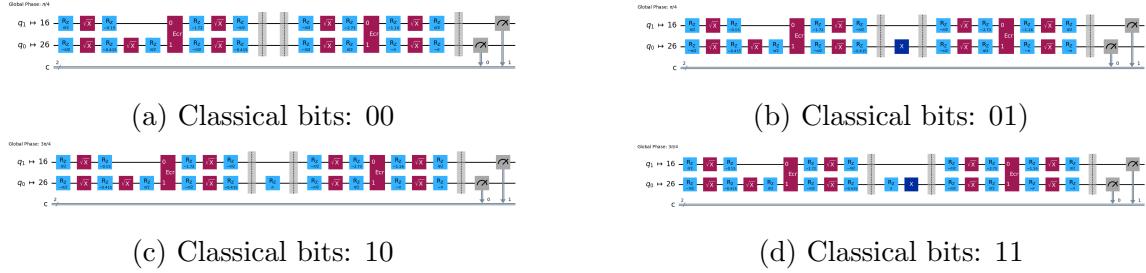


Figure 6.11: Optimized Superdense coding Circuit implementation for transmitting possible classical bits along with their corresponding oracles

two qubits. Thus for execution we will use the Sampler runtime primitive.

```

1 # TO run on local simulator:
2 #1. Use the StatevectorSampler from qiskit.primitives instead
3 sampler=Sampler(backend=backend)
4 sampler.options.default_shots=10_000
5 result_00 = sampler.run([circuit_is_a_00]).result()
6 dist_00 = result_00[0].data.c.get_counts()
7

```

The output of the above code will be the distribution of the measurements on the two qubits. Similar code can be written for all the other circuits by just implementing the required oracle.

4. Step 4: Post-Process Results

Now we plot a histogram of the measurements on the two qubits in the classical format.

```

1 from qiskit.visualization import plot_histogram
2 plot_histogram(dist_00)
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 other possible classical bits.

The following output is produced upon running this code on ibm_brisbane as shown in figure 9.5. 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. Note in the figure you might

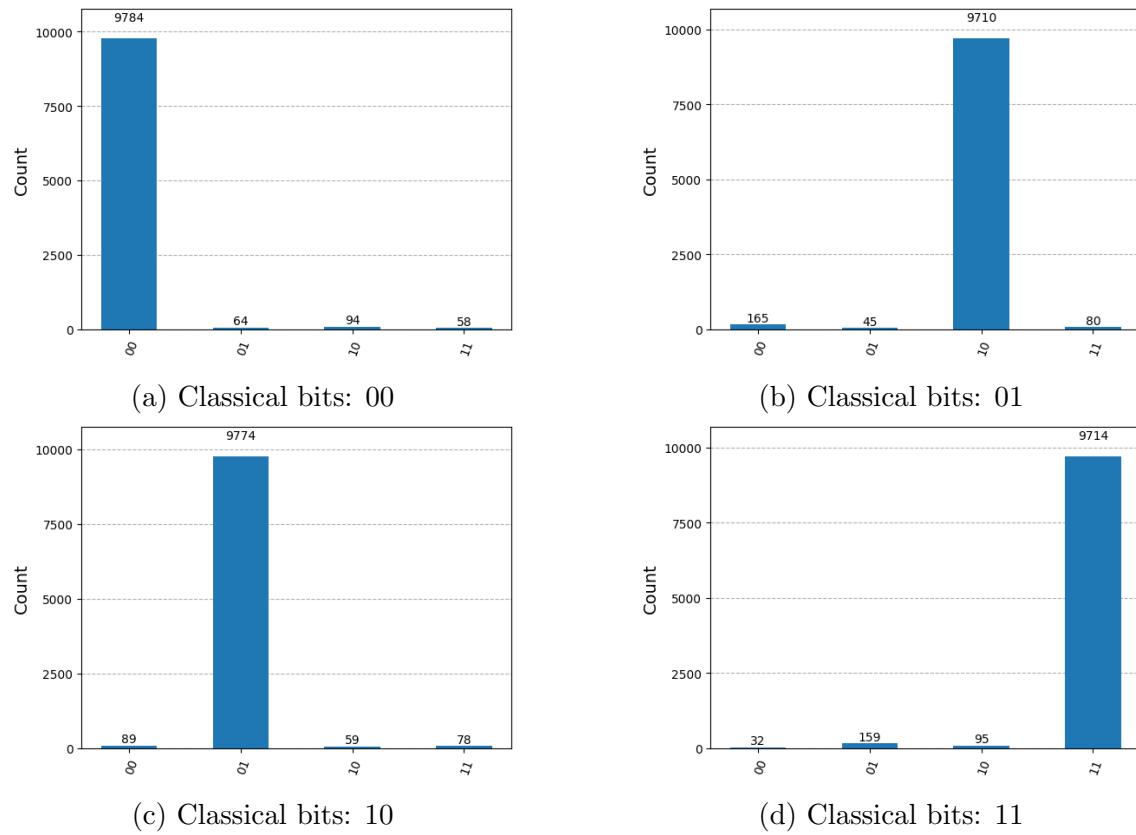


Figure 6.12: Superdense coding Results

see that the results for 01 and 10 are interchanged and one might think this as a mistake but it is not. Qiskit uses little-endian both for classical bit and qubit ordering. So the classical bits are read from right to left. Thus, the results.

Chapter 7

Elitzur-Vaidman Bomb Detection Algorithm

7.1 The Quantum Zeno Effect

Suppose we have a qubit in the state $|0\rangle$ and let's say we want to put it in the $|1\rangle$ state without using unitary transformations. For some small ϵ , we can measure the qubit in a basis that's rotated from $\{|0\rangle, |1\rangle\}$ by an angle ϵ . The probability of getting the qubit moved by ϵ increases as ϵ decreases.

$$\begin{aligned} P(|v\rangle) &= |\langle 0|v \rangle|^2 = |\cos(\epsilon)|^2 \approx 1 - \epsilon^2 \quad |v\rangle = \cos(\epsilon)|0\rangle + \sin(\epsilon)|1\rangle \\ P(|w\rangle) &= |\langle 0|w \rangle|^2 = |\sin(\epsilon)|^2 \approx \epsilon^2 \quad |w\rangle = -\sin(\epsilon)|0\rangle + \cos(\epsilon)|1\rangle \end{aligned}$$

So repeating this process $\approx 1/\epsilon$ times, and rotating the measurement basis by an additional angle ϵ each time, we could slowly drag the qubit $|0\rangle$ to $|1\rangle$. What's the likelihood that we'd ever get a measurement outcome that wasn't the one we wanted? By the union bound, it's of order $(1/\epsilon) \times \epsilon^2 = \epsilon$, so can be made arbitrarily small. This is called the **Quantum Zero Effect**, and one of its discoverers as *Alan Turing*

Another interesting variant of the same kind of effect is called the **Watched Pot Effect**. Say we want to keep a qubit at $|0\rangle$, but it keeps rotation towards $|1\rangle$ (it's drifting). if we keep measuring it in the $\{|0\rangle, |1\rangle\}$ basis every time the qubit has drifted by an angle ϵ , the odds of it jumping to $|1\rangle$ at any given measurement is only ϵ^2 . So if we repeat the measurements $\approx 1/\epsilon$ times, then the probability it ending up at $|1\rangle$ is only $\approx \epsilon$, even though it would have drifted to $|1\rangle$ with certainty had we not measured.

7.2 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:

7.2.1 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.

7.2.2 Approach 2: Quantum Superposition Approach

Here we use a Hadamard Gate before and after the device. The circuit is as shown in the figure 7.1. 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

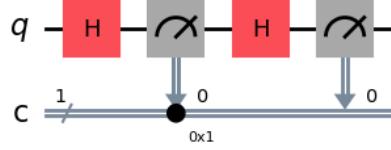


Figure 7.1: Elitzur-Vaidman Bomb Detection Algorithm

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 a 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 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.

7.2.3 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_θ^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 θ is the angle of rotation. The R_θ^Y gate is a rotation gate about the Y-axis. The R_θ^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_θ^Y is a self-inverse gate. $R_\theta^Y = (R_\theta^Y)^\dagger = (R_\theta^Y)^{-1}$. Thus, the action of R_θ^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_θ^Y gate is a rotation gate about the Y-axis by an angle θ . Now clearly, for very small θ , 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

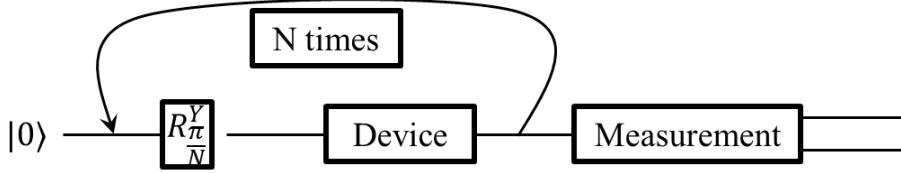


Figure 7.2: Elitzur-Vaidman Bomb Detection Algorithm

figure 7.2. Thus, we pass $|0\rangle$ through the R_θ^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_θ^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 $\left(\frac{\pi}{2N}\right)^{2N}$. As can be seen in the table 7.1, the probability of the

N	Probability of Bomb Exploding
2	0.25
10	7.702×10^{-17}
20	6.105×10^{-45}
\vdots	\vdots
500	1.409×10^{-2503}

Table 7.1: Probability of Bomb Exploding after N iterations

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 8

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.

8.1 The Problem Definition

Definition 8.1.1. 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 8.1. Out of these four possible cases we can see that two are constant and the other two are balanced 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.

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 8.1: Possible Functions

8.2 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 algorithm solving Deutsch's problem with two reversible classical queries looks like as shown in the following classical circuit 8.1.

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

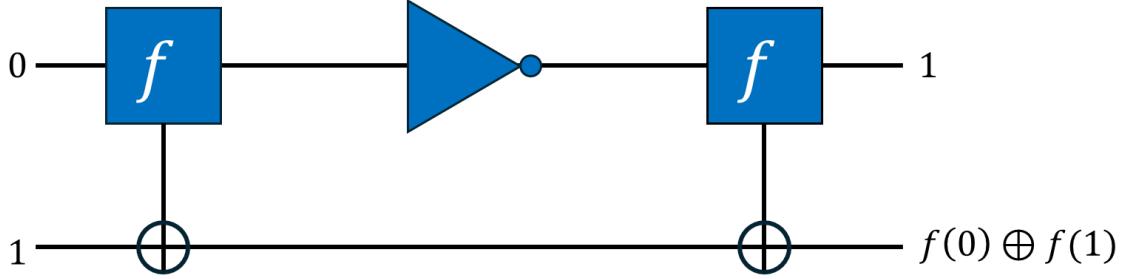


Figure 8.1: Classical Deutsch Circuit

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 8.1.

8.3 The Quantum Solution: Deutsch's Algorithm

8.3.1 Quantum Parallelism

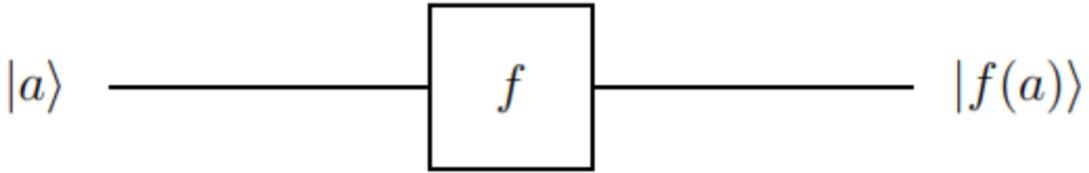
One uniquely quantum-mechanical effect that we can use for building quantum algorithms is quantum parallelism. Suppose we have a classical algorithm that computes some function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$. Then we can build a quantum circuit U (consisting only of Toffoli gates) that maps $|z\rangle |0\rangle \rightarrow |z\rangle |f(z)\rangle$ 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.

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: It can be verified that the corresponding matrix is unitary for and

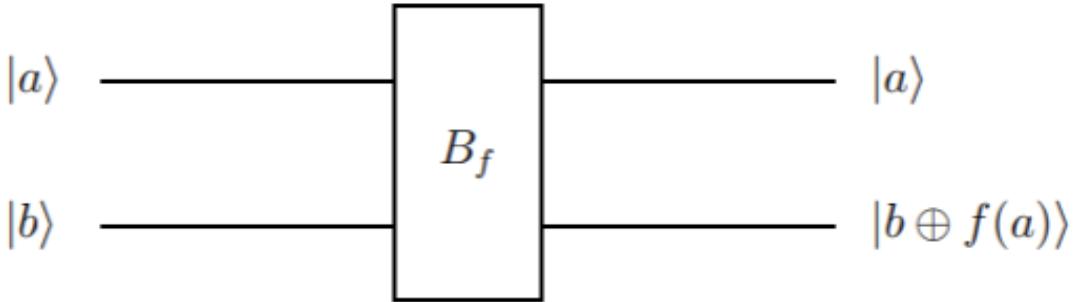


Figure 8.2: Oracle

function f . For example, if $f = f_4$ from the table 8.1, 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

$$B_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$$

(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 8.3. Consider the

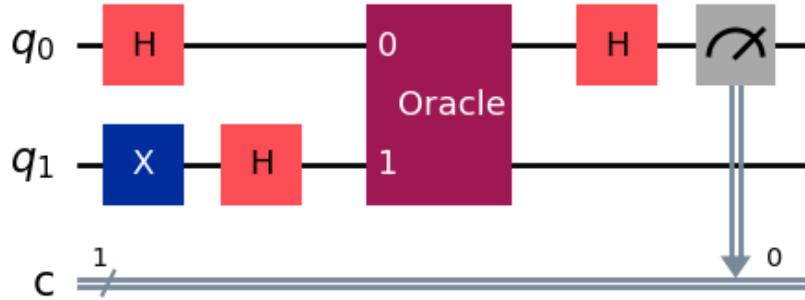


Figure 8.3: Deutsch's Algorithm

oracle as shown in the figure 8.3. 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 .

8.3.2 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 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 result 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 of quantum mechanics and thus quantum computing that allows us to identify the function f in a single query.

8.3.3 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

$$\begin{aligned} & \left(\frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle)\right) \otimes \left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right) \\ &= \left(\frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle)\right) \otimes |- \rangle \\ &= (-1)^{f(0)} \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}(-1)^{f(0)+f(1)}|1\rangle\right) \otimes |- \rangle \end{aligned}$$

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 8.3. We get

$$\begin{aligned} & (H \otimes I) \left(\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}(-1)^{f(0)+f(1)}|1\rangle \right) \otimes (-1)^{f(0)}|-\rangle \right) \\ &= |f(0) \oplus f(1)\rangle \otimes (-1)^{f(0)}|-\rangle \end{aligned}$$

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:

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

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.

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:

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

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.

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 $f(0) \oplus f(1)$.

8.3.4 Example

Example 8.3.1. (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. The goal is to determine which one it is. In other words, given black-box

Function	Input	Output
f_{00}	00	1
	01	0
	10	0
	11	0

(a) f_{00}

Function	Input	Output
f_{01}	00	0
	01	1
	10	0
	11	0

(b) f_{01}

Function	Input	Output
f_{10}	00	0
	01	0
	10	1
	11	0

(c) f_{10}

Function	Input	Output
f_{11}	00	0
	01	0
	10	0
	11	1

(d) f_{11}

Table 8.2: Possible Functions

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,

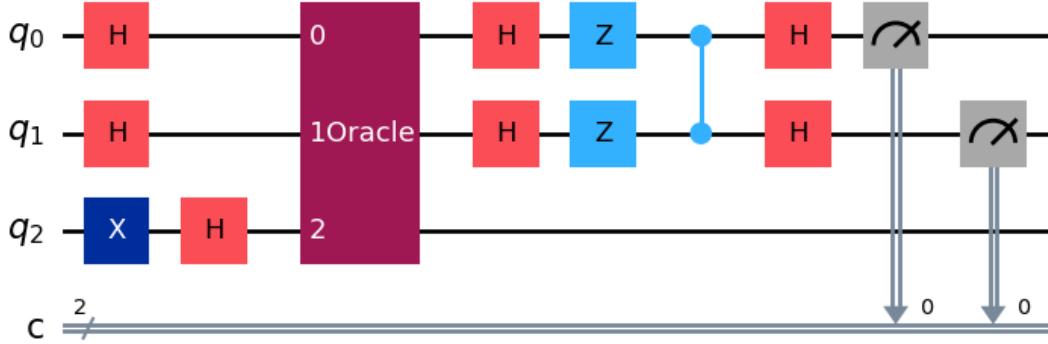


Figure 8.4: Quantum Circuit

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 8.4. The state after the Hadamard transform and just before the query can be written as

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

After the transformation U_f is performed the state will be

$$\left(\frac{1}{2}(-1)^{f_{00}} |00\rangle + \frac{1}{2}(-1)^{f(01)} |01\rangle + \frac{1}{2}(-1)^{f10} |10\rangle + \frac{1}{2}(-1)^{f(11)} |11\rangle \right) \otimes \left(\frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle \right)$$

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 8.4. Note that we have used a phase flip or σ_z gate and a 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 σ_z gate is

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

Example 8.3.2. Implement all the four functions in oracles and simulate the results on a quantum computer.

8.4 Implementation using Qiskit

8.4.1 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.

8.4.2 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

8.4.3 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 8.1. 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: Note that each of the oracle (shown

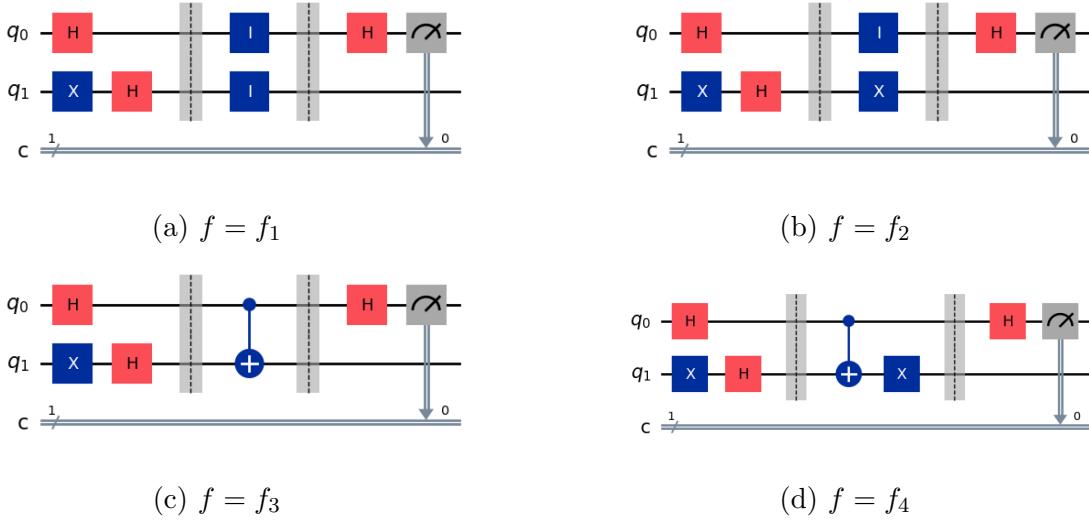


Figure 8.5: Deutsch Circuit implementation for possible functions along with their corresponding oracles

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_isa_1 = pm.run(qc)
9 circuit_isa_1.draw(output="mpl", idle_wires=False, style="iqp")

```

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 8.7. 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. We can see in the figure 8.7a 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.

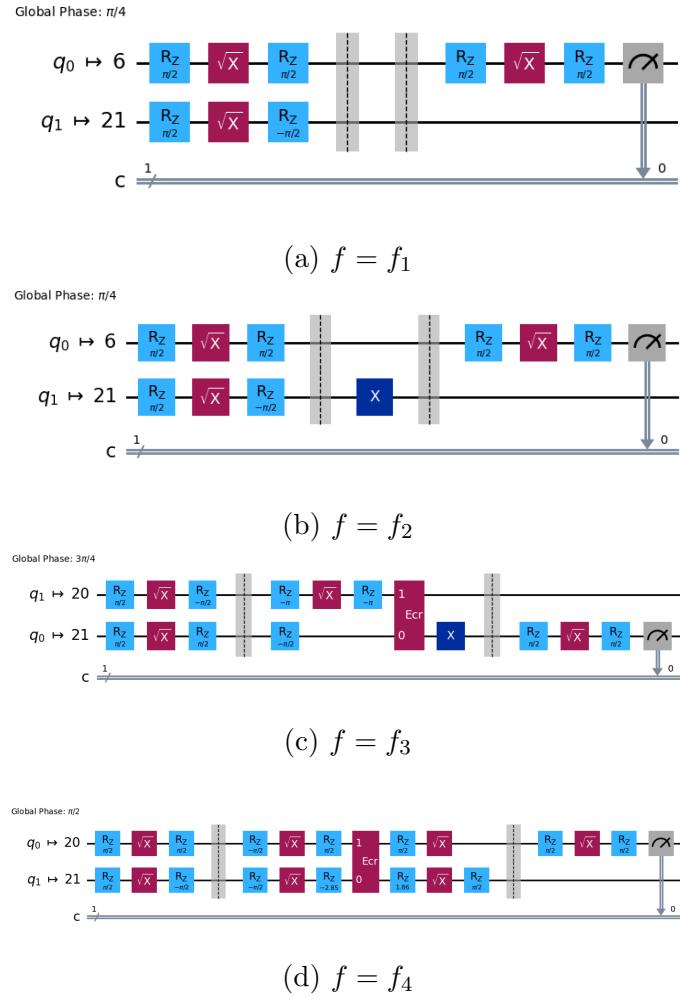


Figure 8.6: Corresponding Optimized Deutsch Circuit

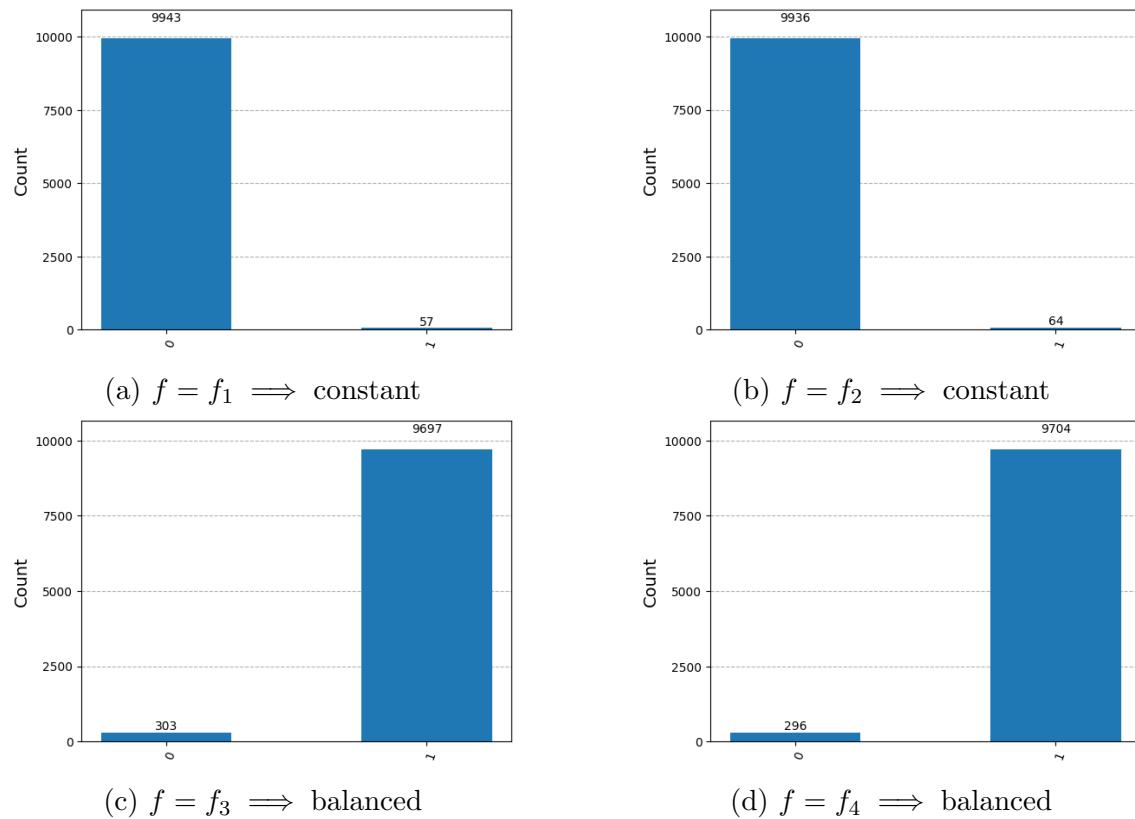


Figure 8.7: Results

Chapter 9

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.

9.1 The Problem Definition

Definition 9.1.1. 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 9.1. 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.

9.2 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

Function	Input	Output
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 9.1: Possible Functions

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 nad it's actually a balanced function. It's only after you've queried in more than half of the spots that you can b 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.

Example 9.2.1. Suppose that the problem is not to distinguish between the constant and balanced functions with certainty, but rather, with some probability of error $\epsilon < 1/2$. What is the performance of the best classical algorithm for this problem.

Recall that a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is said to be balanced if $f(x) = 1$ for exactly half of all possible 2^n values of x .

A single evaluation tells us no information about whether f is constant or balanced, so our success rate/error rate after a single evaluations is $\epsilon = \frac{1}{2}$ (random guessing!). Therefore, consider the case where we do two evaluations. If we obtain two different results, then we immediately conclude that f is balanced. Suppose

instead that we obtain two results that are the same. If f is balanced, then the probability that the first evaluation returns the given result is $\frac{1}{2}$, and the probability that the second evaluation returns the same result is $\frac{2^{n/2}-1}{2^n-1}$ (as there are $2^n/2$ of each result of 0 and 1, 2^n total results, $2^n/2 - 1$ of the given result left after the first evaluation, and $2^n - 1$ total uninvestigated cases after the first evaluation). Therefore, if f is balanced, this occurs with probability $\frac{1}{2} \cdot \frac{2^{n/2}-1}{2^n-1}$, which we can see is less than $\frac{1}{2}$ as:

$$2^n < 2^{n+1} \implies w^n - 2 < 2^{n+1} - 2 \implies \frac{2^n/2 - 1}{2^n - 1} < 1 \implies \frac{1}{2} \frac{2^n/2 - 1}{2^n - 1} < \frac{1}{2}$$

Hence, if we get the same result in two evaluations, we can conclude that f is constant with error $\epsilon < \frac{1}{2}$. We conclude that only 2 evaluations are required for this algorithm.

9.3 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.

9.3.1 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 9.1. 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.

9.3.2 Algorithm

The circuit for Deutsch-Jozsa Algorithm Circuit is as shown in the figure 9.1. The

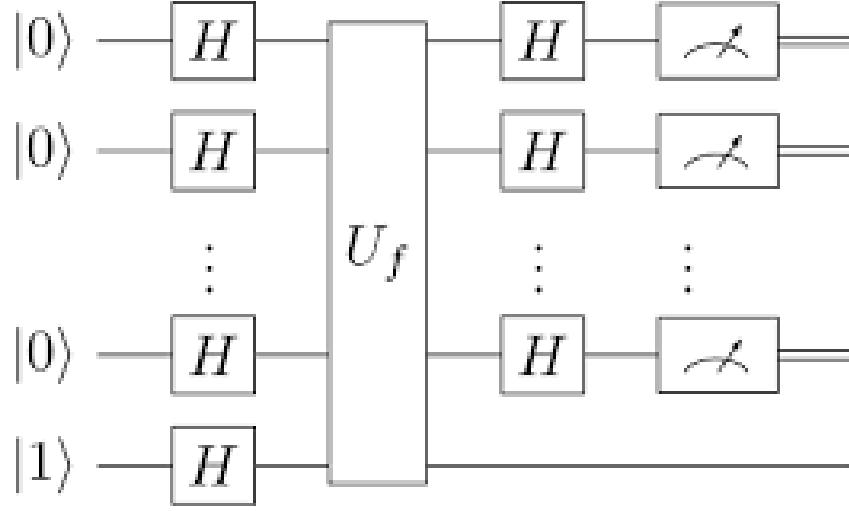


Figure 9.1: Deutsch-Jozsa Circuit

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 3.3.2) 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.

9.3.3 Example

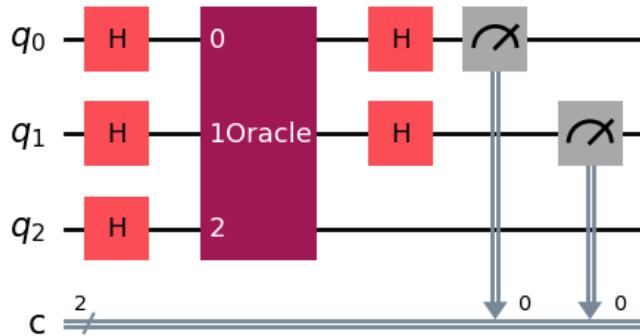


Figure 9.2: Deutsch-jozsa Algorithm for $n=2$

Example 9.3.1. Consider the Deutsch-jozsa algorithm for $n=2$. The possible functions are as shown in the table 9.1 and the circuit for the Deutsch-jozsa algorithm is as shown in the figure 9.2. 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 $|11\rangle$.

9.4 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 9.2: 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.

9.5 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.

In summary, the algorithm is given as follows:

9.6 Implementation using Qiskit

9.6.1 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.

9.6.2 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

Algorithm 1 Deutsch-Jozsa Algorithm

- 1: **Input:** A black box U_f which performs the transformation $|x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle$, for $x \in \{0, \dots, 2^n - 1\}$ and $f(x) \in \{0, 1\}$ and $f(x) \in \{0, 1\}$. It is promised that $f(x)$ is either constant for all values of x , or else $f(x)$ is balanced, that is, equal to 1 for exactly half of all the possible x , and 0 for the other half.
- 2: **Output:** 0 if and only if f is constant.
- 3: **Runtime:** One evaluation of U_f . Always succeeds.
- 4: **Initialize:** $|0^n\rangle|1\rangle$
- 5: Create superposition using Hadamard gates

$$\rightarrow \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

- 6: Calculate function f using U_f

$$\rightarrow \sum_x (-1)^{f(x)} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

- 7: Perform Hadamard transform

$$\rightarrow \sum_z \sum_x \frac{(-1)^{x \cdot z + f(x)}}{\sqrt{2^n}} |z\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

- 8: Measure to obtain the final output $|z\rangle$

- 9: **return** z of the measured state
-

9.6.3 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):

```

```

15 oracle_qc=QuantumRegister(num_qubits+1)
16 oracle=QuantumCircuit(oracle_qc ,name="Oracle")
17 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

```

```

    string")
58         if(len(b_str)!=num_qubits):
59             raise ValueError("Bit string must be of length
60 equal to number of qubits")
61         for qubit in range(num_qubits):
62             if(b_str[qubit]==="1"):
63                 circuit.x(qubits[qubit])
64
65             for qubit in qubits[:-1]:
66                 circuit.cx(qubit,qubits[-1])
67
68             for qubit in range(num_qubits):
69                 if(b_str[qubit]==="1"):
70                     circuit.x(qubits[qubit])
71                 else:
72                     raise ValueError("Oracle type must be 'constant' or
73 balanced'")
74
75             circuit.barrier()
76
77             for qubit in qubits:
78                 circuit.h(qubit)
79
80             for qubit in qubits[:-1]:
81                 circuit.measure(qubit,qubits.index(qubit))
82
83     return circuit
84

```

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 circuit.barrier()
16
17 circuit.h(q0)
18 circuit.h(q1)
19 circuit.measure(q0,0)
20 circuit.measure(q1,1)
21
22 circuit.draw("mpl")
23
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 9.3 according to the table 9.1. 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_manager
2
3 target=backend.target
4 pm = generate_preset_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")
8

```

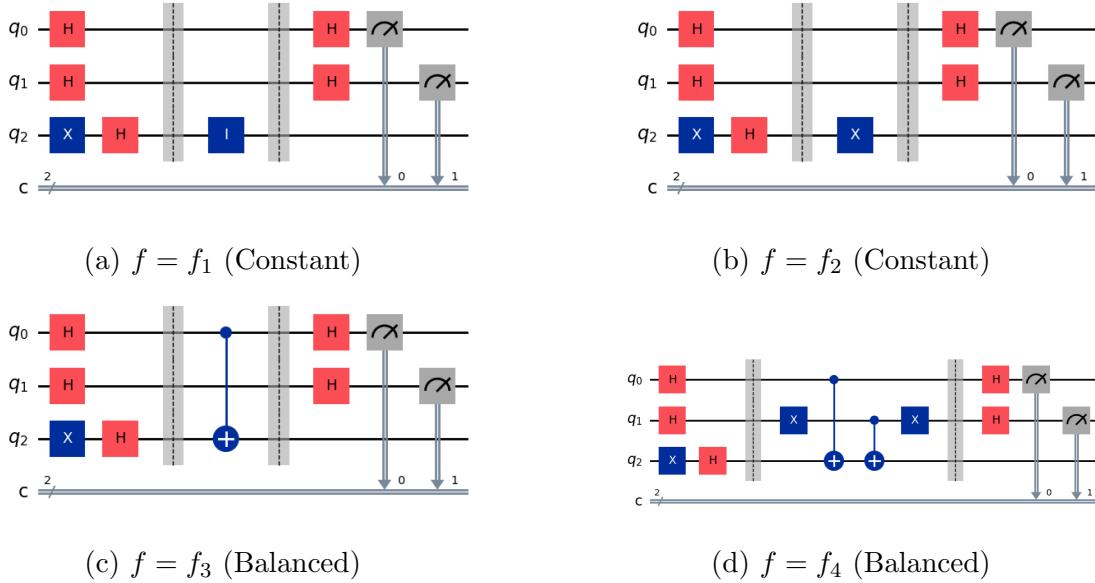


Figure 9.3: Deutsch-Jozsa Circuit implementation for few of the possible functions along with their corresponding oracles

The Optimized circuit shown in figure 9.4 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

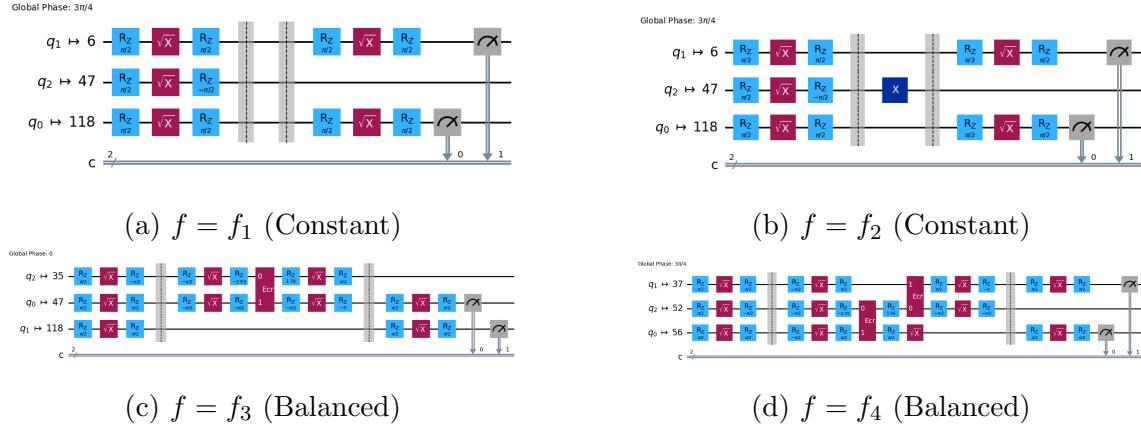


Figure 9.4: Optimized Deutsch-Jozsa Circuit implementation for possible functions along with their corresponding oracles

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 9.5. 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. We can see in the figure 9.5a 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

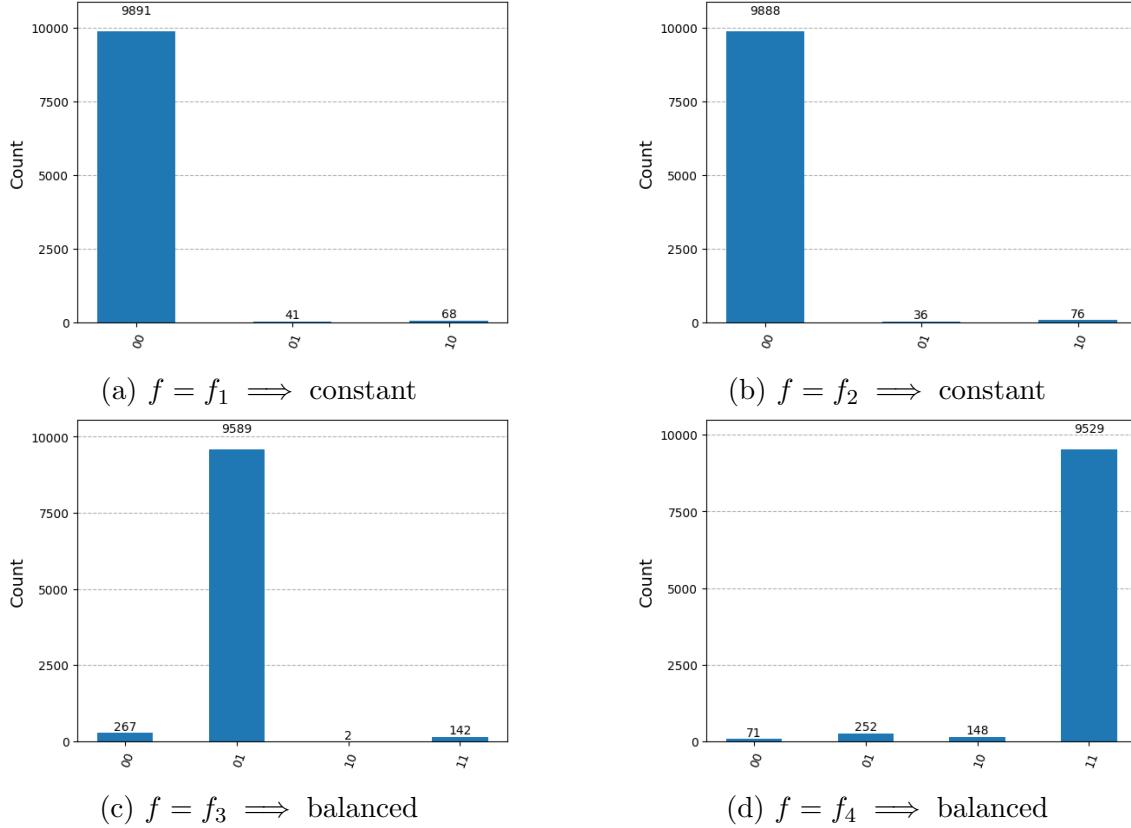


Figure 9.5: Deutsch-Jozsa Results

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 9.6. The corresponding optimal circuit is shown in figure 9.7. The result is shown in figure 9.8.

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

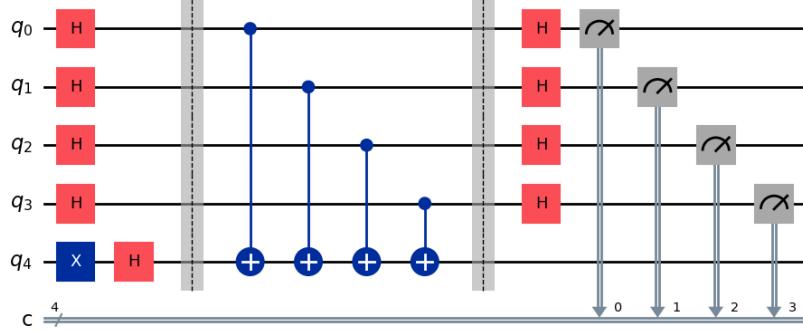


Figure 9.6: Four Qubit Example of Balanced Function

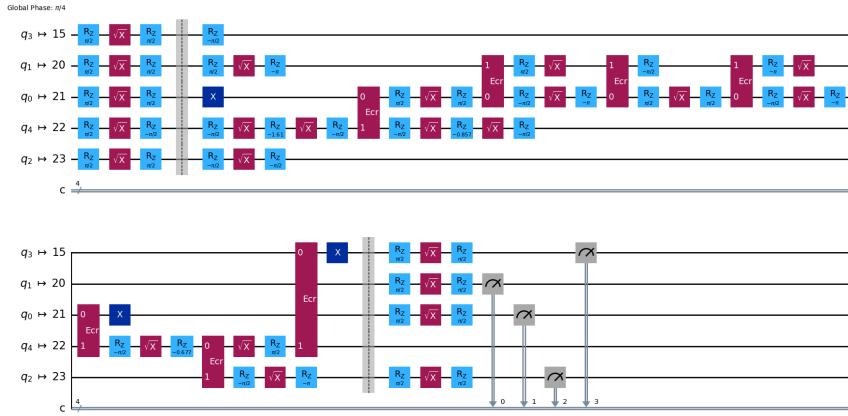


Figure 9.7: Optimized Deutsch-Jozsa Circuit for 4 qubit balanced function

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 9.9 and the figure 9.10 are equivalent figures where in one case the function implemented is hidden inside the oracle.

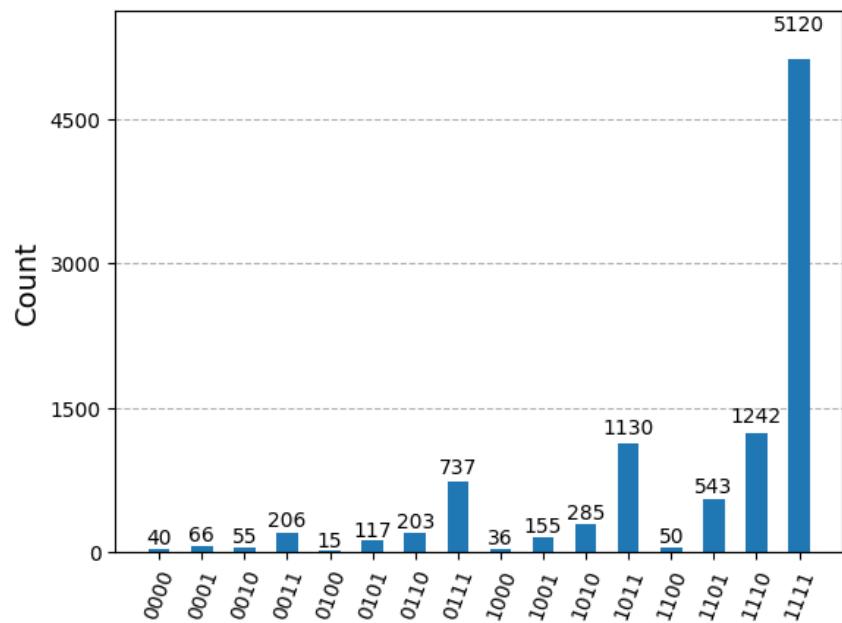


Figure 9.8: Deutsch-Jozsa results for 4 qubit balanced function

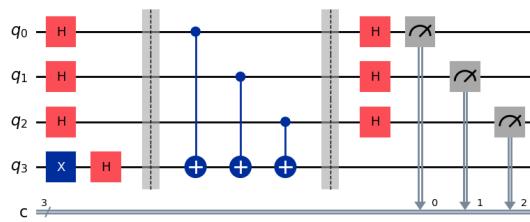


Figure 9.9: Deutsch-Jozsa Example of 3 qubit balanced function

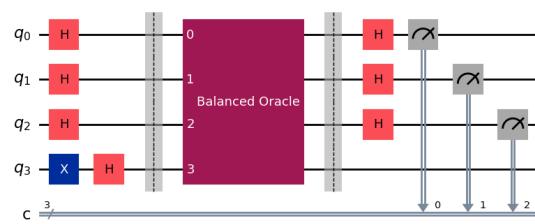


Figure 9.10: Deutsch-Jozsa Corresponding 3 qubit oracle for balanced function

Chapter 10

Bernstein-Vazirani Algorithm

Now we will discuss the Bernstein-Vazirani problem. It is also called the Fourier sampling problem, although there are more general formulations of this problem that also go by that name.

In order to describe this problem, it will be helpful to introduce some notation. For two binary strings $x = x_{n-1} \dots x_0$ and $y = y_{n-1} \dots y_0$, we define

$$x \cdot y = x_{n-1}y_{n-1} \oplus \dots \oplus x_0y_0$$

We'll refer to this operation as the binary dot product. An alternative way to define it as follows

$$x \cdot y = \begin{cases} 1, & \text{if } x_{n-1}y_{n-1} + \dots + x_0y_0 \text{ is odd} \\ 0, & \text{if } x_{n-1}y_{n-1} + \dots + x_0y_0 \text{ is even} \end{cases}$$

Notice that this is a symmetric operation, meaning that the result doesn't change if we swap x and y , so we're free to do that whenever it's convenient.

One way to think about the binary product $x \cdot y$ is that it equals the parity of those qubits of x in positions where the string y has a 1, which is equivalent to the parity of those bits of y in positions where the string x has a 1.

With this notation in hand we can now define the Bernstein-Vazirani problem.

10.1 The Problem

Input: A function $f : \{0,1\}^n \rightarrow \{0,1\}$ such that there exists a binary string $s = s_{n-1} \dots s_0$ for which $f(x) = s \cdot x$ for all $x \in \{0,1\}^n$.

Output: The string s .

We don't actually need a new algorithm for this problem, the Deutsch-Jozsa algorithm (not including the post-processing step of computing the OR of the measurement outcomes) solves it. In the interest of clarity, let's refer to the quantum circuit above (without the classical post-processing step) as the Deutsch-Jozsa circuit.

10.2 The Classical solution

While the Deutsch-Jozsa circuit solves the Bernstein-Vazirani problem with a single query, any classical query algorithm must take at least n queries to solve the problem. This can be reasoned through a so-called information theoretic argument. Each classical query reveals a single bit of information about the solutions, and there are n bits of information that need to be uncovered.

It is, in fact possible to solve the Bernstein-Vazirani problem classically by querying the function on each of the n strings having a single 1, in each possible position, and 0 for all other bits, which reveals the bits of s one at a time.

So, the advantage of quantum over classical algorithms for this problem is 1 query versus n queries.

10.3 Quantum solution

To analyze the Deutsch-Jozsa circuit for the Bernstein-Vazirani problem, we'll begin with a quick observation. Using the binary dot product, we can alternatively describe the action of n Hadamard gates on the standard basis states of n qubits as follows.

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

Similar to what we saw when analyzing Deutsch's algorithm, this is because the value $(-1)^k$ depends only on whether k is even or odd.

Turning to the circuit, after the first layer of the Hadamard gates is performed, the state of the $n + 1$ qubits is

$$|-\rangle \otimes \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$$

The query gate is then performed, which (through the phase kickback phenomenon) transforms the state to

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

Using the formula above for the action of the second layer of Hadamard gates, the state becomes

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

We can now make some simplifications to this states, focusing on the exponent of -1 inside the sum. We're promised that $f(x) = s \cdot x$ for some string $s = s_{n-1} \dots s_0$ so we can express the state as

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

Because $s \cdot x$ and $x \cdot y$ are binary values, we can replace the addition with the exclusive OR again because the only thing that matters for an integer in the exponent of -1 is whether it is even or odd. Making use of the symmetry of the binary dot product, we obtain this expression for the state:

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

(Parentheses have been added for clarity, although they aren't really necessary: it is convention to treat the binary dot product as having higher precedence than the exclusive-OR. An easy way to remember this is that the binary dot product looks like multiplication and the exclusive-OR looks like addition.)

At this point we'll make use of the following formula.

$$(s \cdot x) \oplus (y \cdot x) = (s \oplus y) \cdot x$$

We can obtain the formula through a similar formula for bits,

$$(ac) \oplus (bc) = (a \oplus b)c$$

together with an expansion of the binary dot product and bitwise exclusive-OR:

$$\begin{aligned} (s \cdot x) \oplus (y \cdot x) &= (s_{n-1}x_{n-1}) \oplus \dots \oplus (s_0x_0) \oplus (y_{n-1}x_{n-1}) \oplus \dots \oplus (y_0x_0) \\ &= (s_{n-1} \oplus y_{n-1})x_{n-1} \oplus \dots \oplus (s_0 \oplus y_0)x_0 \\ &= (s \oplus y) \cdot x \end{aligned}$$

This allows us to express the state of the circuit immediately prior to the measurements like this:

$$|-\rangle \oplus \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \sum_{y \in \{0,1\}^n} (-1)^{(s \oplus y) \cdot x} |y\rangle$$

The final step is to make use of yet another formula, which works for every binary string $z = z_{n-1} \dots z_0$.

$$\frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{z \cdot x} = \begin{cases} 1, & \text{if } z = 0^n \\ 0, & \text{if } z \neq 0^n \end{cases}$$

(Here we're using a simple notation for strings: 0^n is the all-zero string of length n)

A simple way to argue that this formula works is to consider the two cases separately. If $z = 0^n$, then $z \cdot x = 0$ for every string $x \in \{0,1\}^n$, so the value of each term in the sum is 1, and we obtain 1 by summing and dividing by 2^n . On the other hand, if any one of the bits of z is equal to 1, then the binary product $z \cdot x$ is equal to 0 for exactly half of the possible choices for $x \in \{0,1\}^n$ and 1 for the other half - because the value of the binary dot product $z \cdot x$ flip (from 0 to 1 or from 1 to 0) if we flip the bit the bit of x in any position where z has a 1.

If we now apply this formula to simplify the state of the circuit prior to the measurements, we obtain

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

owing to the fact that $s \oplus y = 0^n$ if and only if $y = s$.

Thus, the measurement reveals precisely the string s we are looking for.

10.4 Implementation

Chapter 11

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. Also the quantum algorithm for this problem introduces some powerful techniques in a simple form. It's interesting for the following two reasons.

1. It improves on the progression of black-box problems where quantum algorithms outperform classical algorithms. The quantum algorithm requires exponentially fewer queries than even probabilistic classical algorithms that can err with constant probability, say $\frac{1}{4}$.
2. The quantum algorithm for Simon's problem introduces a technique that transcends the black-box model. When looked at the right way, the ideas introduced in Simon's algorithm lead naturally to Shor's algorithm for the discrete log problem - which is not a black-box model problem! Shor discovered his algorithms (for discrete log and factoring) soon after seeing Simon's algorithm, and in his paper, he acknowledges that we was inspired by Simon's algorithm.

11.1 The Problem Definition

Definition 11.1.1. Given a function $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ which takes n input bits and returns n output bits (for a positive integer n). The function is a two-to-one function (which means that, for every point in the range, there are exactly two pre-images. In other words, for every value that the function attains, there are

exactly two points a and b , in the domain that both map to that value.), such that $f(x) = f(x \oplus s)$ for $s \in \{0, 1\}^n$ (s is a non-zero bit string). We are required to find s . Similar to Deutsch-Jozsa problem, the function f is promised to obey a certain property, although the property is slightly more complicated than before.

$$f(x) = f(y) \iff [x \oplus y \in \{0^n, s\}]$$

Here \oplus is the bitwise direct sum modulo 2 (bitwise XOR, you take the XOR of the first bit of a with the first bit of b , and then you take XOR of the second bit of a with the second bit of b , and so on). For example, $1101 \oplus 0111 = 1010$ or $01 \oplus 01 = 00$. Such a pair of points where $a \neq b$ and $f(a) = f(b)$ then a and b are colliding pair. Note that for every colliding pair (a, b) , the $a \oplus b = s$.

Example 11.1.1. 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,

$$\begin{aligned} f(000) &= f(000 \oplus 111) = f(111); & f(001) &= f(001 \oplus 111) = f(110) \\ f(010) &= f(010 \oplus 111) = f(101); & f(011) &= f(011 \oplus 111) = f(100) \\ f(100) &= f(100 \oplus 111) = f(011); & f(101) &= f(101 \oplus 111) = f(010) \\ f(110) &= f(110 \oplus 111) = f(001); & f(111) &= f(111 \oplus 111) = f(000) \end{aligned}$$

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 function output in tabular form: 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 11.1 that the function is a two-to-one function and the function is periodic with period $s = 111$, thus following Simon's property as well. 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 . Note that if you take any colliding pair (a, b) and then $a \oplus b$ is always the same 3-bit string. For example, take 001 and 110 which is a colliding pair since, $a = 001 \neq b = 110$ and $f(a = 001) = f(b = 110) = 000$, then

Input	Output
000	111
001	000
010	110
011	010
100	010
101	110
110	000
111	111

Table 11.1: Example of Simon’s Algorithm

$a \oplus b = 001 \oplus 110 = s = 111$. Note that, for an arbitrary two-to-one function, the bitwise XORs can be different for different colliding pairs. So the two-to-one functions that satisfy the Simon property are special ones, for which these bit-wise XORs are always the same.

11.2 The Classical Solution

On a Classical Computer: One way to solve this is to find a colliding pair. But notice that any two distinct n -bit strings could be a colliding pair for some s . So if we make a query at some point a , we learn the value of $f(a)$, but we have no idea for which $b \neq a$, you get a colliding pair. For example, if we have a budget of m queries, query f at m random points in $\{0, 1\}^n$ and then check if there’s a collision among them. If any two are a colliding pair then their bitwise XOR is the answer s . If there are no collisions then the algorithm is out of luck; in that case it fails to find r . How large the m should be set to so that the probability of a collision is at least $3/4$. Since there are 2^n points in the domain, and each pair of queries will collide with probability of $\frac{1}{2^n}$. Since there are around m^2 pairs, the expected number of collisions is roughly m^2 times $\frac{1}{2^n}$. Setting $m = \sqrt{2^n}$ suffices to make this expectation constant. You may recognize in this analysis the so-called “birthday paradox”, where you consider what the changes are that there are two people in a group of say 23 people who have the same birthday. It’s 50% assuming that people’s birthdays are uniformly distributed. 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. So, once we have a colliding pair, it's easy to find s by taking the bitwise XOR of the colliding pair. 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 $x \oplus x = 0$).

11.2.1 Classical Lower Bound

Theorem 11.2.1. *Any classical algorithm that solves the Simon's problem with worst-case success probability at least $\frac{3}{4}$ must make $\Omega(\sqrt{2^n})$ queries.*

We cannot deduce this simply based in the fact that this is the best algorithm that we could come up with. How can we be sure that there isn't some clever trick that we haven't discovered?. Also, notice that f is not an arbitrary two-to-one function. It is a two-to-one function with Simon property, which is a very special structure. So it's conceivable that a classical algorithm can somehow take advantage of that structure to find a collision in an unusual way.

The proof to the theorem is not as trivial and requires proof. Thus, now we shall begin with proof to this theorem.

Proof. This proof uses some standard techniques that arise in computational complexity, however, this account assumes no prior background in the area.

The first part of the proof is to “play the adversary” by coming up with a way of generating an instance of f that will be hard for any algorithm. Note that picking some fixed f will not work very well. A fixed f has a fixed r associated with it and the first two queries of the algorithm could be 0^n and r , which would reveal r to the algorithm after only two queries. rather, we shall randomly generate instances of f . First, we pick r at random, uniformly from $\{0, 1\}^n - 0^n$. Picking r does not fully specify f but it partitions $\{0, 1\}^n$ into 2^{n-1} colliding pairs of the form $\{x, x \oplus r\}$ in the lexicographic order. Let T be the set of all such representatives: $T = \{s : s = \min\{x, x \oplus r\} \text{ for some } x \in \{0, 1\}^n\}$. Then we can define f in terms of a random one-to-one function $\phi : T \rightarrow \{0, 1\}^n$ uniformly over all the $2^n(2^n - 1)(2^n - 2) \dots (2^n - 2^{n-1} + 1)$ possibilities. The definition of f can then be taken as

$$f(x) = \begin{cases} \phi(x) & \text{if } x \in T \\ \phi(x \oplus r) & \text{if } x \notin T \end{cases}$$

We shall prove that no classical probabilistic algorithm can succeed with probability $\frac{3}{4}$ on such instances unless it makes a very large number of queries.

The next part of the proof is to show that, with respect to the above distribution among inputs, we need only consider deterministic algorithms (by which we mean ones that make no probabilistic choices). The idea is that any probabilistic algorithm is just a probability distribution over all the deterministic algorithms, so its success probability p is the average of the success probabilities of all the deterministic algorithms (where the average is weighted by the probabilities). At least one deterministic algorithm must have success probability $\geq p$ (otherwise the average would be less than p). Therefore (because we have a fixed probability distribution of the input instances), we need only consider deterministic algorithms.

Next, consider some deterministic algorithm and the first query that it makes: $(x_1, y_1) \in \{0, 1\}^n \times \{0, 1\}^n$, where x_1 is the input to the query and y_1 is the output of the query. The result of this will just be a uniformly random element of $\{0, 1\}^n$, independent of r . Therefore the first query by itself contains absolutely no information about r .

Now consider the second query (x_2, y_2) (without the loss of generality, we can assume that the inputs to all queries are different; otherwise the redundant queries could be eliminated from the algorithm). There are two possibilities: $x_1 \oplus x_2 = r$ (collision) or $x_1 \oplus x_2 \neq r$ (no collision). But the first case arises with probability only $\frac{1}{2^n - 1}$. With probability $1 - \frac{1}{2^n - 1}$, we are in the second case, and all the algorithm deduces about r is that $r \neq x_1 \oplus x_2$ (it has ruled out just one possibility among $2^n - 1$).

We continue our analysis of the process by induction on the number of queries. Suppose that $k - 1$ queries, $(x_1, y_1), \dots, (x_{k-1}, y_{k-1})$ have been made without any collisions so far. (No collision so far means that, for all $1 \leq i < j \leq k - 1$, $y_i \neq y_j$). Then all that has been deduced about r is that it is not $x_i \oplus x_j$ for all $1 \leq i < j \leq k - 1$. In other words, up to $(k - 1)(k - 2)/2$ possibilities of r have been eliminated. When the next query (x_k, y_k) is made, the number of potential collisions arising from it are at most $k - 1$ (there are $k - 1$ previously made queries to collide with). Therefore the probability of collision at query k is at most

$$\frac{k - 1}{2^n - 1 - (k - 1)(k - 2)/2} \leq \frac{2k}{2^{n+1} - k^2}$$

Since the collision probability bound in the above equation holds for all k , the probability of a collision occurring somewhere among m queries is at most the sum of the

right side of equation with k varying from 1 to m :

$$\sum_{k=1}^m \frac{2k}{2^{n+1} - k^2} \leq \sum_{k=1}^m \frac{2m}{2^{n+1} - m^2} \leq \frac{2m^2}{2^{n+1} - m^2}$$

If this quantity is to be at least $\frac{3}{4}$ then

$$\frac{2m^2}{2^{n+1} - k^2} \geq \frac{3}{4}$$

When we solve form in the above inequality, we get,

$$m \geq \sqrt{\frac{6}{11}} 2^n$$

which gives the desired bound.

Actually, there is a slight technicality remaining. We have shown that $\sqrt{(6/11)2^n}$ queries are necessary to attain a collision with probability $\frac{3}{4}$; whereas the algorithm is not technically required to make the queries that include a collision. Rather, the algorithm is just require to deduce r , and it is conceivable that an algorithm could deduce r some other way without collision occurring. But any algorithm that deduces r can be modified so that it makes one additional query that collides with a previous one. Hence, we have a slightly smaller bound of $\sqrt{(6/11)2^n} - 1$, but this is still $\Omega(\sqrt{2^n})$. \square

11.3 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 and the some classical post-processing. The circuit for Simon's Algorithm is as shown in the figure 11.1. Note that U_f gate has different (but very similar) form from before because the input and output of the function f are n-bit strings:

$$U_f |x\rangle |y\rangle = |x\rangle |f(x) \oplus y\rangle$$

where \oplus denotes the bitwise XOR.

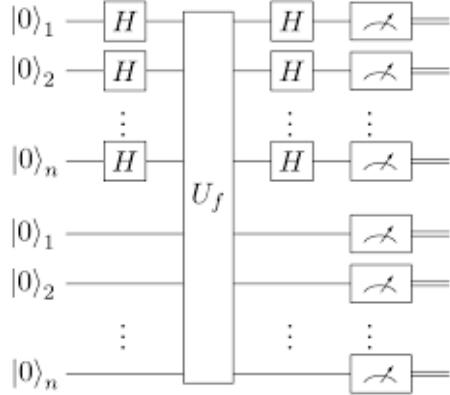


Figure 11.1: Simon's Algorithm

11.3.1 Algorithm

Before going into the classical post-processing, let us look at the quantum part.

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 first n input bits 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:

$$U_f \left(\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |0\rangle^{\otimes n} \right) = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle$$

Consider the special case when $s = 0^n$, which means that f is a one-to-one function. After applying the hadamard transform on the first n qubits, we can write from the

above as

$$\begin{aligned}
(H^{\otimes n} \otimes I^{\otimes n}) \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle &= \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} H^{\otimes n} |x\rangle |f(x)\rangle \\
&= \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle |f(x)\rangle \\
&= \sum_{y \in \{0,1\}^n} |y\rangle \left(\frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{x \cdot y} |f(x)\rangle \right)
\end{aligned}$$

so the probability that the measurement results in each string y is

$$\left\| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{x \cdot y} |f(x)\rangle \right\|^2 = \frac{1}{2^n}$$

Thus, the outcomes is simply a uniformly distributed n bit string when $s = 0^n$. The case where $s \neq 0^n$ is more interesting case. The analysis from before still works to imply that the probability to measure any given string y is

$$\left\| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{x \cdot y} |f(x)\rangle \right\|^2$$

Let $A = \text{range}(f)$. If $z \in A$, there must exists 2 distinct strings $x_z, x'_z \in \{0,1\}^n$ such that

$$f(x_z) = f(x'_z) = z$$

and moreover it is necessary that $x_z \oplus x'_z = s$ (which is equivalent to $x'_z = x_z \oplus s$). Now,

$$\begin{aligned}
\left\| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{x \cdot y} |f(x)\rangle \right\|^2 &= \frac{1}{2^n} \sum_{z \in A} (({}_1^{x_z \cdot y} + (-1)^{x'_z \cdot y}) |z\rangle) \|z\|^2 \\
&= \left\| \frac{1}{2^n} ((-1)^{x_z \cdot y} + (-1)^{(x_z \oplus s) \cdot y}) |z\rangle \right\|^2 \\
&= \left\| \frac{1}{2^n} \sum_{z \in A} (-1)^{x_z \cdot y} (1 + (-1)^{s \cdot y}) |z\rangle \right\|^2 \\
&= \begin{cases} 2^{-(n-1)} & \text{if } s \cdot y = 0 \\ 0 & \text{if } s \cdot y = 1 \end{cases}
\end{aligned}$$

In this calculation we have used the fact that

$$(x_z \oplus s) \cdot y = (x_z \cdot y) \oplus (s \cdot y)$$

So, the measurement always results in some string y that satisfies $s \cdot y = 0$, and the distribution is uniform over all of the strings that satisfy this constraint.

It's sometimes said that what makes quantum computer so powerful is that they can perform several computations at the same time. but this view is misleading. The state was obtained by making just one single query, and it appears to contain information about all the values of the function f . however, it's not possible to extract more than one value of the function from this state. In particular, if we measure this state in the computational basis then what we end up with is just one pair $(a, f(a))$, where a is sampled from the uniform distribution. And you don't need any quantum devices to generate such a sample. You could just flip a coin n times to create a random a and then perform one query to get $f(a)$. So instead of measuring in the computational basis, we can instead - via a unitary operation - measure with respect to another basis. In some cases, for a well-chosen basis, we can acquire information about some global property of f (instead of the value of f at some specific point).

Now we try to find out some useful measurement basis for the case f satisfies the Simon property. The inputs to the function partition into colliding pairs. Note that this time there is no phase kickback as there was in the previous algorithms. 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. Thus the state of the last n qubits collapses to some value $f(z)$ and the residual state of the first n qubits is a uniform superposition of the pre-image of that value. 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 if we could somehow extract both z and $z \oplus s$ from this state then we could denote the value of s (by taking XOR, $a \oplus (a \oplus s) = s$). But we can only measure the state once, after which its state collapses.

If we measure in the computational basis we just get either a or $a \oplus s$, neither of which is sufficient to learn anything about the value of s . We can think of measuring in the computational basis this way: we are getting a random m -bit string, which is

devoid of any information about the structure of f . So, if we want to make progress than we should definitely not measure the first n qubits in the computational basis.

Something remarkable happens if we apply a Hadamard transform (this is due to Fourier sampling) to each of the n qubits before measuring in the computational basis. We can calculate this as shown. 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 what happens when we measure this state in the computational basis. Notice that, for each $y \in \{0, 1\}^n$. 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.

Classical Post Processing

When we run the circuit above, in the special case when $s = 0^n$, the measurement results in each string $y \in \{0, 1\}^n$ with probability

$$p_y = \frac{1}{2^n}$$

otherwise, in the case that $s \neq 0^n$ the probability to obtain each string y is

$$p_y = \begin{cases} 2^{-(n-1)} & \text{if } s \cdot y = 0 \\ 0 & \text{if } s \cdot y = 1 \end{cases}$$

Thus, in both the cases, the measurement results in some string y that satisfies $s \cdot y = 0$, and the distribution is uniform over all of the strings that satisfy this constraint.

In other words, the outcome of measurement is a uniformly distributed random y in the orthogonal component of s (that is, for which $y \cdot s = 0$). Thus, the superposition will consist of only those values of y for which $y \cdot s = 0$. This does not produce enough information for us to deduce r , but it reveals partial information about s . Namely the bits of s satisfy the linear equation

$$y_1 s_1 + y_2 s_2 + \dots + y_n s_n = 0 \pmod{2}$$

Each execution of the procedure produces an independent random $y \in \{0, 1\}^n$ that is orthogonal to s (in the sense that $y \cdot s = 0$). Suppose that we repeat the process $n - 1$ times (so the number of f -queries is $n - 1$). Then combining the resulting y 's we obtain a system of $n - 1$ linear equations in (mod 2 arithmetic). Note that this is not exactly the type of system of linear equations we are used to in linear algebra, but the system can be efficiently solved using similar methods.

There are exactly n linearly independent values of y such that $y \cdot s = 0$, and one of these is the trivial solution $y = 0$. Therefore, there are $n - 1$ non-trivial, linearly independent solutions to $y \cdot s = 0$. But if y_1 and y_2 are linearly independent solutions, $(y_1 + y_2) \cdot s = y_1 \cdot s + y_2 \cdot s = 0$. So linear combinations of solutions are also solutions. This gives us a total of 2^{n-1} y 's such that $y \cdot s = 0$. To solve for s , we need to find exactly $n - 1$ non-trivial, linearly independent y such that $y \cdot s = 0$. To solve for s , we need to find exactly $n - 1$ non-trivial, linearly independent y such that $y \cdot s = 0$. Thus, it's enough information to determine s provided we repeat the process several times and accept a small probability of failure. You will get a unique non-zero solution s if you are lucky and y_1, \dots, y_{n-1} are linearly independent.

$$\begin{bmatrix} y_{1,1} & y_{1,2} & \dots & y_{1,n} \\ y_{2,1} & y_{2,2} & \dots & y_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n-1,1} & y_{n-1,2} & \dots & y_{n-1,n} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

If you have linear independence, solve the system to get a candidate solution $s' \neq 0^n$, and test that $f(0^n) = f(s')$. If $f(0^n) = f(s')$, you must know that $s = s'$ and the problem has been solved. If $f(0^n) \neq f(s')$, it must be the case that $s = 0^n$ (because if this were not so, the unique nonzero solution to the linear equations would have been s). Either way, once you have linear independence, you can solve the problem.

11.3.2 Example

Example 11.3.1. Consider the function $f : \{0, 1\}^4 \rightarrow \{0, 1\}^4$ as shown in the table 11.2. Here, $s = 1001$. Now, say the above shown function has been implemented in a

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 11.2: Function

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 11.2 we get, since $n=4$ we will get 16 terms:

$$\begin{aligned} & \frac{1}{4} [|0000\rangle |1111\rangle + |0001\rangle |0001\rangle + |0010\rangle |1110\rangle + |0011\rangle |1101\rangle + |0100\rangle |0000\rangle + |0101\rangle |0101\rangle \\ & \quad |0110\rangle |1010\rangle + |0111\rangle |1001\rangle + |1000\rangle |0001\rangle + |1001\rangle |1111\rangle + |1010\rangle |1101\rangle + |1011\rangle |1110\rangle \\ & \quad + |1100\rangle |0101\rangle + |1101\rangle |0000\rangle + |1110\rangle |1001\rangle + |1111\rangle |1010\rangle] \end{aligned}$$

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.

11.4 Complexity Analysis

If each rows of a $n-1 \times n$ matrix is an independent sample from the set of $y \in \{0, 1\}^n$ such that $y \cdot s = 0$, then the probability that the matrix has rank $n-1$ is at least $1/4$.

$$\prod_{k=1}^{\infty} \left(1 - \frac{1}{2^k}\right) = 0.288788 > \frac{1}{4}$$

So we now have a quantum algorithm that makes $n-1$ queries in all and succeeds in finding s is at least $\frac{1}{4}$. This fails with probability at most $\frac{3}{4}$. It's easy to reduce the failure probability to $\left(\frac{3}{4}\right)^5 < \frac{1}{4}$ by repeating the procedure five times. Now, repeat the process $4m$ times. The probability of not finding a linearly independent set during one of the iterations is less than

$$\left(1 - \frac{1}{4}\right)^{4m} < e^{-m}$$

For example, if $m = 10$ this probability is less than $1/20000$.

Thus, for any constant $\epsilon > 0$, Simon's algorithm can solve the problem we are consider with the error probability at most ϵ using $O(n)$ queries to the black-box. In conclusion, $\Omega(\sqrt{2^n})$ queries are necessary for any classical algorithm to attain success probability $\frac{3}{4}$, whereas order $\mathcal{O}(n)$ queries are sufficient for a quantum algorithm to attain success probability $\frac{3}{4}$.

11.5 Conclusion

Simon's problem is a black-box problem that was specially designed to be very hard for probabilistic classical algorithms and easy for quantum algorithms - thereby improving on previous classical vs quantum query cost separations.

Problem	Quantum	Classical Deterministic	Classical Probabilistic
Simon	$O(n^2)$	$O(2^{n/2})$	$O(2^{n/2})$

Table 11.3: Complexity Analysis

But Simon's problem doesn't immediately look like a problem that one would care about in the real world. When Simon's work first came out, people were wondering what to make of it. Although it provided a very strong classical vs quantum query cost separation, it looked like a contrived problem. Moreover, a contrived black-box problem, which is not even a conventional computing problem, involving input data.

This may be one's first impression, but there's actually more to it than that. Look again at the Simon property: for all x , $f(x) = f(x \oplus s)$. This is kind of like a periodicity property of a function, which would be written as: for all x , $f(x) = f(x + s)$. And periodic functions arise in many contexts. We'll soon see that variations of Simon's problem in this direction are very fruitful.

Chapter 12

Discrete log Problem

The quantum “algorithms” that we’ve seen up until now are not algorithms in usual sense. They are in the black-box model, where one is given an unknown function and the goal is to extract information about the functions with as few queries to the function as possible. Often, the unknown function is promised to have a special kind of structure (such as the function arising in Simon’s problem).

In the next section I will described a remarkable algorithm for solving the discrete log problem (DLP). This is not a black box problem! It is a conventional computational problem where the input is given as a binary string and the output is also a binary string. No classical polynomial algorithm is known for this problem. In fact, the presumed hardness of this problem has been the basis of crypto systems (such as the Diffie-Hellman key exchange protocol). In 1994, Peter Shor discovered a polynomial algorithm for this problem, and it’s based on the underlying methodologies used in Simon’s algorithm - which may sound surprising, since Simon’s problem is a black-box problem.

In this section, I define the discrete log problme. In the next section I will describe Shor’s polynomial quantum algorithm for this problem. In order to define the discrete log problem, we first need to be familiar with two sets, called \mathbb{Z}_m and \mathbb{Z}_m^* .

12.1 Definition of \mathbb{Z}_m and \mathbb{Z}_m^*

Definition 12.1.1. (of the ring \mathbb{Z}_m) For any positive integer $m \geq 2$, define \mathbb{Z}_m as the set $\{0, 1, \dots, m - 1\}$, equipped with addition and multiplication modulo m.

A familiar example of a ring is the set of all integer \mathbb{Z} equipped with “ordinary”

addition and multiplication. Note that there are only two elements of \mathbb{Z} that have multiplicative inverses (namely $+1$ and -1). What are the elements of \mathbb{Z}_m that have multiplicative inverses? It depends on m . Let's look at the case where $m = 9$. The elements of \mathbb{Z}_9 that have multiplicative inverses are $1, 2, 4, 5, 7$ and 8 ($0, 3$ and 6 do not have multiplicative inverses). Here, 2 and 5 are multiplicative inverses for each other since, $2 \times 5 = 10(\text{mod}9) = 1$. Similarly, $4 \times 7 = 28(\text{mod}1) = 1$, $1 \times 1 = 1(\text{mod}9) = 1$, $8 \times 8 = 64(\text{mod}9)$

Definition 12.1.2. (of the group \mathbb{Z}_m^*). For any positive integer $m \geq 2$, the set \mathbb{Z}_m^* consists of all elements of \mathbb{Z}_m that have multiplicative inverses. \mathbb{Z}_m^* is a group.

For the purposes of DLP, we need only consider \mathbb{Z}_p^* for prime p . In that case, it's known that every non-zero element of \mathbb{Z}_p^* has a multiplicative inverse. Therefore we can use this simple definition of the group \mathbb{Z}_p^* for case where p is prime.

Definition 12.1.3. (of the group \mathbb{Z}_p^*). For any prime p , define \mathbb{Z}_p^* as the group with elements $\{1, \dots, p-1\}$, where the operations is multiplication modulo p .

12.2 Generators of \mathbb{Z}_p^* and the exponential/log functions

An element g of the group \mathbb{Z}_p^* is called a generator of the group if the set of all powers g is the group. Whenever a group has such an element, it is called cyclic.

Let's consider the case where $p = 7$ as an example. $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$. Obviously 1 is not a generator, since all powers of 1 are just 1 . What about 2 ? 2 is not a generator either, because if we list the powers of 2 , which are $2^0, 2^1, 2^2$, and so on, we get the sequence, $1, 2, 4, 1, 2, 4, \dots$ so we only get the set $\{1, 2, 4\}$, which is a proper subgroup of \mathbb{Z}_7^* . What about 3 ? 3 is a generator, since the powers of 3 are $1, 2, 3, 6, 4, 5, 1, 3, 2, \dots$, which is the entire set of \mathbb{Z}_7^* . It turns out that \mathbb{Z}_p^* is cyclic for any prime p .

Theorem 12.2.1. For any prime p , there exists $g \in \mathbb{Z}_p^*$ such that

$$\mathbb{Z}_p^* = \{g^k : k \in \{0, 1, \dots, p-2\}\}$$

Notice that the exponents run from 0 to $p-2$. This makes sense because the size of \mathbb{Z}_p^* is $p-1$ (it's not p because $0 \neq \mathbb{Z}_p^*$). So the set of exponents of a generator is \mathbb{Z}_{p-1} (it's not \mathbb{Z}_p). And, if the elements of \mathbb{Z}_p^* are expressed as powers of a generator g , then $g^x g^y = g^{x+y \text{mod} p-1}$ holds for any $x, y \in \mathbb{Z}_{p-1}$. In other words, multiplication in \mathbb{Z}_p^* corresponds between the additive group \mathbb{Z}_{p-1} and the multiplicative group \mathbb{Z}_p^* .

Definition 12.2.1. (discrete exp function). Relative to a prime modulus p and a generator g of \mathbb{Z}_p^* , let's first define the discrete exponential function $\exp_g : \mathbb{Z}_{p-1} \rightarrow \mathbb{Z}_p^*$ as, for all $r \in \mathbb{Z}_{p-1}$,

$$\exp_g(r) = g^r$$

Definition 12.2.2. (discrete log function). Relative to a prime modulus p and a generator g of \mathbb{Z}_p^* , define the discrete log function $\log_g : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_{p-1}$ as the inverse of the discrete exponential function \exp_g . The input to \log_g is some $s \in \mathbb{Z}_p^*$, and the output is the value of $r \in \mathbb{Z}_{p-1}$ such that $g^r = s$.

12.3 Discrete exponential problem

For the discrete exp problem, the input is (p, g, r) , where

- p is an n-bit prime
- g is a generator of \mathbb{Z}_p^*
- $r \in \mathbb{Z}_{p-1}$

And the output is: $\exp_g(r)$, which is g^r .

How hard is it to compute this function? of course, g^r is equal to g multiplied by itself r times. So $\exp_g(r)$ can obviously computed by r multiplications (the precise number is actually $r - 1$). But r is an n -bit number, so r can be roughly as large as 2^n . That's an exponentially large number of multiplication operations!. Using this approach, the circuit size would be exponential in n . But there's a simple trick for doing this more efficiently, called repeated squaring trick.

12.3.1 Repeated Squaring Trick

The idea is the following. You multiply g by itself to get g^2 . Then you multiply g^2 by itself to get g^4 . And then you multiply g^4 by itself to get g^8 , and so on. This way you can compute g^{2n} at the cost of only n multiplications. That's how the repeated squaring trick works when the exponent r is a power of 2.

What if r is not a power of 2? Then above idea can be adjusted to compute any n -bit exponent r with fewer than $2n$ multiplications. The idea is based on the fact that g^r can be written as

$$g^{r_n \dots r_3 r_2 r_1} = ((\dots (g^{r_n})^2 \dots g^{r_3})^2 g^{r_2})^2 g^{r_1}$$

where $r_n \dots r_3 r_2 r_1$ is the binary representation of an n -bit exponent r .

Note that $O(n)$ multiplications, at cost $O(n \log n)$ gates each, leads to a classical gate cost of order $O(n^2 \log n)$ for computing the discrete exponential problem.

12.4 Discrete log problem

For the discrete log problem (DLP), the input is (p, g, s) , where

- p is an n bit prime.
- g is a generator of \mathbb{Z}_p^* .
- $s \in \mathbb{Z}_p^*$

And the output is $\log_g(s)$, which is the $r \in \mathbb{Z}_{p-1}$ for which $g^r = s$.

12.5 Shor's Algorithm for the discrete log problem

The overall idea behind Shor's algorithm is to convert the discrete log problem (DLP) into a generalization of Simon's problem, and then to solve the generalization of Simon's problem. Since DLP is not a black-box problem, how can such a conversion work? It works by creating a function with a property similar to Simon's that can be efficiently implemented so as to simulate black-box queries to the function. This is Shor's function.

12.5.1 Shor's function with a property similar to Simon's

Recall that an instance of the discrete log problem consists of three n -bit numbers: p (an n -bit prime), g (a generator of \mathbb{Z}_p^*), and s (an element of \mathbb{Z}_p^*). Shor's function $f : \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1} \rightarrow \mathbb{Z}_p^*$ is defined as

$$f(a_1, a_2) = g^{a_1} s^{a_2}$$

for all $a_1, a_2 \in \mathbb{Z}_{p-1}$. What's interesting about this function is where the collisions are. When is $f(a_1, a_2) = f(b_1, b_2)$?

Theorem 12.5.1. *For an instance (p, g, s) of DLP, the function $f : \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1} \rightarrow \mathbb{Z}_p^*$ defined by equation above has the property that*

$$f(a_1, a_2) = f(b_1, b_2) \quad \text{if and only if } (a_1, a_2) - (b_1, b_2) \text{ is a multiple of } (r, 1)$$

where $r = \log_g(s)$.

The significance of this is that it resembles the Simon property. It's the analogue of the Simon property when we switch from mod 2 arithmetic to mod $p-1$ arithmetic. To see this, recall that the Simon property can be stated as: $f(a) = f(b)$ if and only if $a \oplus b$ is either a string of n zeroes or the string r . Notice that: $a \oplus b$ is the same as $a - b$ in the mod 2 arithmetic. So we can write the Simon property as:

Simon property for $f : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2^n$: $f(a_1, \dots, a_n) = f(b_1, \dots, b_n)$ if and only if $(a_1, \dots, a_n) - (b_1, \dots, b_n)$ is a multiple of (r_1, \dots, r_n) in mod 2 arithmetic.

and here's again is the property that Shor's function has: **Property of Shor's function for** $f : \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1} \rightarrow \mathbb{Z}_p^*$: $f(a_1, a_2) = f(b_1, b_2)$ if and only if $(a_1, a_2) - (b_1, b_2)$ is a multiple of $(r, 1)$ in mod $p-1$ arithmetic.

Proof. The proof is elementary, but we'll go through it carefully. Although we do not know what r is, we do know that an r exists such that $s = g^r$. This means that $s^{a_2} = g^{ra_2}$. Therefore, $f(a_1, a_2) = g^{a_1}g^{-ra_2} = g^{a_1-ra_2}$. It's nice that to write f this way, because then

$$\begin{aligned} f(a_1, a_2) &= f(b_1, b_2) \\ \text{if and only if } a_1 - ra_2 &= b_1 - rb_2 \\ \text{if and only if } (a_1, a_2) \cdot (1, -r) &= (b_1, b_2) \cdot (1, -r) \\ \text{if and only if } ((a_1, a_2) - (b_1, b_2)) \cdot (1, -r) &= 0 \end{aligned}$$

In this equation, the dot product being zero is like an orthogonality relation between the vector $(a_1, a_2) - (b_1, b_2)$ and the vector $(1, -r)$. here's a schematic sketch of the vector $(1, -r)$.

Notice that the vector $(r, 1)$ is orthogonal to the vector $(1, -r)$ in that their dot product is zero. Now, in a two-dimensional space, the vectors that are orthogonal to a particular vector are all multiples of one of the orthogonal vectors. So, we might expect $(a_1, a_2) - (b_1, b_2)$ to be a multiple of $(r, 1)$. This intuition (valid for vector spaces with inner products) is confirmed in our context by a simple calculation:

$$\begin{aligned} (v_1, v_2) \cdot (1 - r) &= 0 \\ \text{if and only if } v_1 &= rv_2 \\ \text{if and only if } v_2 &= k \text{ and } v_1 = rk, \text{ for some } k \in \mathbb{Z}_{p-1} \\ \text{if and only if } (v_1, v_2) &= k(r, 1), \text{ for some } k \in \mathbb{Z}_{p-1} \end{aligned}$$

Therefore, $f(a_1, a_2) = f(b_1, b_2)$ if and only if $f(a_1, a_2) - f(b_1, b_2)$ is a multiple of $(r, 1)$. \square

Chapter 13

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 for 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 measurement 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 Hadamard 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, ω . The Quantum Fourier transform is a key ingredient for quantum factoring and many other interesting algorithms. It is an efficient quantum algorithm for performing a Fourier transform of quantum mechanical amplitudes. But one important task which it does enable is phase estimation, the approximation of the eigenvalues of an unitary operator under certain circumstances. This allows us to solve several other interesting problems, including the order-finding problem and the factoring problem . Phase estimation can also be combined with quantum search algorithm to solve the problem of counting solutions to a search problem. It can also be used to solve the hidden subgroup problem ,a generalization of the phase estimation and order-finding problem that has among its special cases an efficient quantum algorithm for the dis-

crete logarithm problem, another problem though to be intractable on a classical computer.

13.1 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 $0 \leq k \leq N - 1$ and they all lie in the Complex Plane \mathbb{C} . The nth roots of unity are equally spaced around the unit circle in the complex plane (it can be seen that ω lies on the units circle so $|\omega| = 1$, and the angle made by the root ω is the angle $\phi = 2\pi/n$, and squaring ω , we double the angle. Thus, raising to the jth power, ω^j has a phase angle $\phi = 2\pi j/N$ and is still an Nth root of unity.). **Properties:**

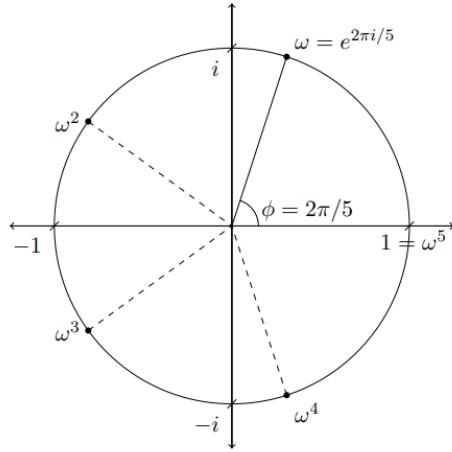


Figure 13.1: The 5th Complex roots of 1

- $\omega_N^k = \omega_N^{(k \bmod N)}$ 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, $n = 4$.

- Sum of all these powers of ω , we get zero: $1 + \omega + \omega^2 + \dots + \omega^{N-1} = 0$.

$$1 + \omega^k + \omega^{2k} + \dots + \omega^{(N-1)k} = \sum_{j=0}^{n-1} \omega^{kj} = 0$$

for all $k \in \{1, 2, \dots, N-1\}$, since $\omega^N = 1$, it's obvious that $\sum_{j=0}^{N-1} \omega^{Nj} = N$.

13.2 Classical Fast Fourier Transform

The FFT was a major breakthrough for classical computers. Because the Fourier transform is an $N \times N$ matrix, straightforward multiplication by F_N takes $\mathcal{O}(N^2)$ steps 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 is incredibly important in signal processing that essentially all of the electronics rely on it. Without the FFT, modern electronics would have far fewer capabilities and would be much slower than they are today. The FFT required only that $N = 2^n$ for some integer n , but this is a relatively easy requirement because 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 again 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 into even and odd columns. Consider the following figure 13.2: In the first frame, we have represented the whole Fourier transform matrix, by describing the j th row and k th column: ω^{jk} . In the next frame, we separate the odd an 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 $\omega^{n/2} = -1$).

$$\begin{array}{c}
 \begin{array}{c|c}
 k & \\
 \hline
 j & \omega^{jk} \\
 \vdots & \\
 \alpha_0 & \\
 \alpha_1 & \\
 \alpha_2 & \\
 \alpha_3 & \\
 \alpha_4 & \\
 \vdots & \\
 \alpha_{n-1} &
 \end{array}
 \end{array}
 = \begin{array}{c}
 \begin{array}{c|c}
 2k & 2k+1 \\
 \hline
 j & \omega^{2jk} \quad \omega^j \omega^{2jk} \\
 \vdots & \\
 \alpha_0 & \\
 \alpha_2 & \\
 \vdots & \\
 \alpha_{n-2} & \\
 \hline
 \alpha_1 & \\
 \alpha_3 & \\
 \vdots & \\
 \alpha_{n-1} &
 \end{array}
 \end{array}
 = \begin{array}{c}
 \begin{array}{c|c}
 2k & 2k+1 \\
 \hline
 j & \omega^{2jk} \quad \omega^j \omega^{2jk} \\
 \hline
 j + \frac{N}{2} & \omega^{2jk} \quad -\omega^j \omega^{2jk} \\
 \vdots & \\
 \alpha_0 & \\
 \alpha_2 & \\
 \vdots & \\
 \alpha_{n-2} & \\
 \hline
 \alpha_1 & \\
 \alpha_3 & \\
 \vdots & \\
 \alpha_{n-1} &
 \end{array}
 \end{array}$$

even columns odd columns

Figure 13.2: FFT of size 8

Notice that both the odd side and even side contain the term ω^{2jk} . But if ω is the primitive N th root of unity, then ω^2 is the primitive $N/2$ nd root of unity. Therefore, the matrices whose j,k th entry is ω^{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 $\hat{f}(j)$.

$$\hat{f}(j) = (\vec{F}_{2^n/2} \vec{f}_{\text{even}})(j) + \omega^j (\vec{F}_{2^n/2} \vec{f}_{\text{odd}})(j)$$

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 till $QFT_1 = 1$. Thus, the FFT takes $\mathcal{O}(N \log N)$ steps to compute the Fourier transform of a function of size N . This greatly simplifies our calculation.

13.3 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 ω_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, $F_N^\dagger F_N = I$.

Example 13.3.1. Transform $|\psi\rangle = \frac{|0\rangle + |3\rangle}{\sqrt{2}}$ using F_4 .

$$F_4 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \iota & -1 & \iota \\ 1 & -1 & 1 & -1 \\ 1 & \iota & -1 & \iota \end{pmatrix}$$

$$|\psi\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$F_4 |\psi\rangle = \frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \iota & -1 & \iota \\ 1 & -1 & 1 & -1 \\ 1 & \iota & -1 & \iota \end{pmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{2\sqrt{2}} \begin{bmatrix} 2 \\ 1-\iota \\ 0 \\ 1+\iota \end{bmatrix}$$

$$= \frac{2|0\rangle + (1-\iota)|1\rangle + (1+\iota)|3\rangle}{2\sqrt{2}}$$

Example 13.3.2. 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 $N = 4, \omega_4 = e^{\frac{2\pi\iota}{4}} = i$. Thus, the QFT matrix of size 4 is given by:

$$QFT_4 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \iota & -1 & -\iota \\ 1 & -1 & 1 & -1 \\ 1 & -\iota & -1 & \iota \end{pmatrix}$$

which is clearly equal to $H^{\otimes 2}$.

Example 13.3.3. 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:

$$QFT_4 |f\rangle = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \iota & -1 & -\iota \\ 1 & -1 & 1 & -1 \\ 1 & -\iota & -1 & \iota \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 4 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$QFT_4 |g\rangle = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \iota & -1 & -\iota \\ 1 & -1 & 1 & -1 \\ 1 & -\iota & -1 & \iota \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix}$$

$$QFT_4 |h\rangle = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \iota & -1 & -\iota \\ 1 & -1 & 1 & -1 \\ 1 & -\iota & -1 & \iota \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ \iota \\ -1 \\ -\iota \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.5\iota \\ -0.5 \\ -0.5\iota \end{pmatrix}$$

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 Fourier transforms of $|g\rangle$ and $|h\rangle$ is a difference of relative 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 α 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.

13.3.1 Properties of QFT

1. 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 i th and j th 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

2. **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 $|\hat{f}(x)\rangle$. 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 β_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 β_k by ω_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 13.3.4. 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.

3. 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 13.3. In general, the wider the range of

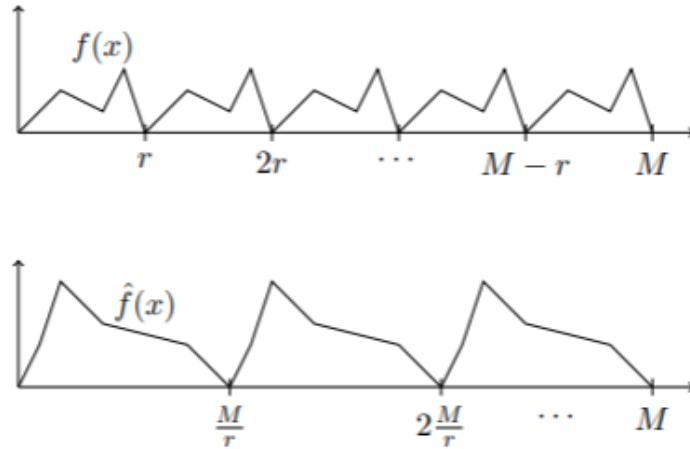


Figure 13.3: Period and Wavelength Relationship

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 13.4 and 13.5 ($M = 2^n$). Suppose $|\alpha\rangle =$

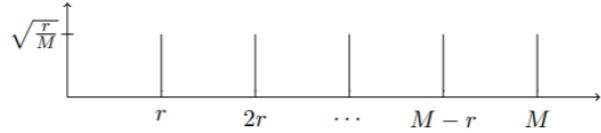


Figure 13.4: $f(x)$

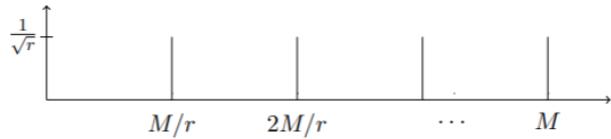


Figure 13.5: QFT of $f(x)$

$\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{2^n-1} \end{pmatrix}$ 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 transform 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'Hopital's rule.

$$\lim_{x \rightarrow \frac{k2^n}{r}} \frac{1 - \omega^{x2^n}}{1 - \omega^{rx}} = \lim_{x \rightarrow \frac{k2^n}{r}} \frac{-2^n \omega^{x2^n-1}}{-r \omega^{rx-1}} = \lim_{x \rightarrow \frac{k2^n}{r}} \frac{2^n \omega^{x2^n}}{r \omega^{rx}} = \frac{2^n}{r}$$

Plugging 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 proceed in general, imagine a periodic functions in r of this type as a basis for any periodic function. Allow the possibility of relative phase shifts and we can thus prove this property in general. \square

13.4 Quantum Fourier Transform with quantum gates

One of the most useful ways of solving a problem in mathematics of computer science is to transform it into some other problem for which a solution is known. Fourier transform is one of the few transforms which appear so often that it is studied for their own sake. A great discovery of quantum computation has been that some such transformation can be computed much faster on a quantum computer than on a

classical computer which has enabled construction of fast algorithms for quantum computers. 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 of 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 implementation of 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 ω^j . Consider the figure 13.6. In quantum algorithm, the first step

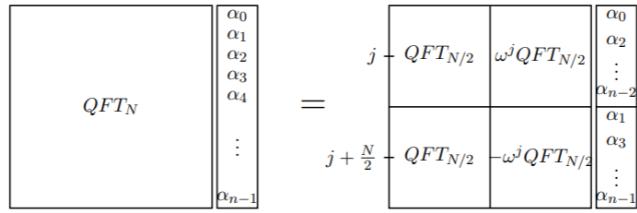


Figure 13.6: circuit

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 Hadamard to the least significant bit as shown in the figure 13.7.

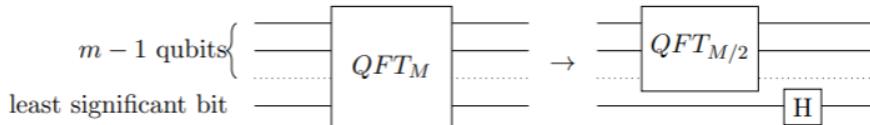


Figure 13.7: circuit

Now to carry out the phase multiplication, we need to multiply each odd term j by the phase ω_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 ω^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 Hadamard transform, QFT_N has been reduced to $QFT_{N/2}$ as shown in the figure 13.8.

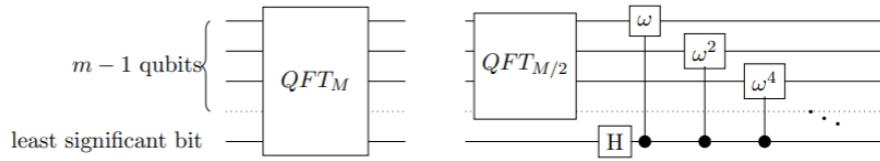


Figure 13.8: circuit

QFT_N is reduced to $QFT_{N/2}$ and N additional gates

Example 13.4.1. Let us construct 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_ϕ . **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 13.9.

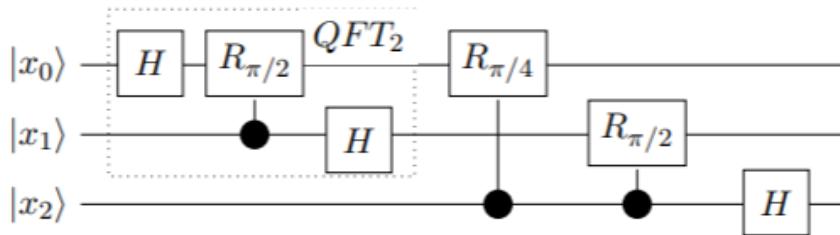


Figure 13.9: circuit

13.5 Efficient implementation of the Quantum Fourier Transform

How can quantum Fourier transform may be implemented by quantum circuits. The quantum Fourier transform on an orthogonal basis $|0\rangle, \dots, |N-1\rangle$ is defined to be

a linear operation with the following action on the basis states. Recall that

$$QFT_{2^m} |j\rangle = \frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i j k / 2^m} |k\rangle$$

Equivalently, the action on an arbitrary state may be written

$$\sum_{j=0}^{N-1} x_j |j\rangle \rightarrow \sum_{k=0}^{N-1} y_k |k\rangle$$

where the amplitudes y_k are the discrete Fourier transform of the amplitudes x_j . It is not obvious from the definition, but this transformation is a unitary transformation, and thus can be implemented as the dynamics for a quantum computer.

The transform takes as input a vector of complex numbers $|k\rangle$ where the length N of the vector is a fixed parameter. It outputs the transformed data, a vector of complex numbers $|j\rangle$. The quantum Fourier transform is exactly the same transformation, although the conventional notation for the quantum Fourier transform is somewhat different. 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} , then an efficient implementation of QFT_2 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 QFT_2 is just a fancy name for a Hadamard transformation:

$$\tilde{QFT}_2 |j\rangle = \frac{1}{\sqrt{2}} \sum_{k=0}^1 \omega_2^{jk} |k\rangle = \frac{1}{\sqrt{2}} (|0\rangle + \omega_2^j |1\rangle) = \frac{1}{\sqrt{2}} (|0\rangle + (-1)^j |1\rangle)$$

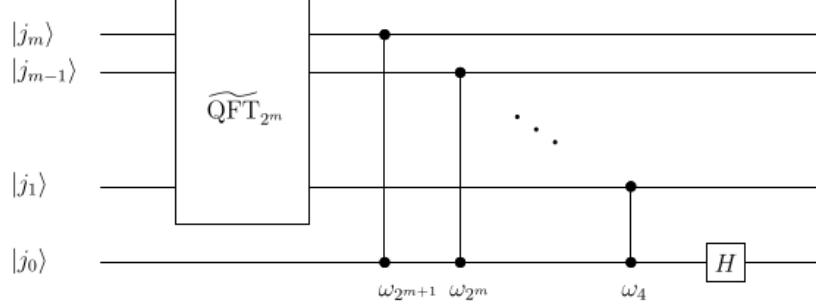


Figure 13.10: QFT circuit

For general case $m \geq 2$, the following circuit computes $\tilde{QFT}_{2^{m+1}}$:

This diagram assume you know how to implement the transformation \tilde{QFT}_{2^m} , but using the fact that \tilde{QFT}_2 is the same as Hadamard 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

$$j' = j_m j_{m-1} \dots j_1 = \lfloor j/2 \rfloor$$

$$k' = k_{m-1} k_{m-2} \dots k_0 = k - k_m 2^m$$

The initial state $|j\rangle$ may therefore be written $|j'\rangle |j_0\rangle$, and the operation \tilde{QFT}_2^m maps this state to

$$\frac{1}{\sqrt{2^m}} \sum_{k'=0}^{2^m-1} \omega_{2^m}^{j'k'} |k'_0 k'_1 \dots k'_{m-1}\rangle |j_0\rangle$$

The controlled phase-shifts then transform this state to

$$\frac{1}{\sqrt{2^m}} \sum_{k'=0}^{2^m-1} \omega_{2^m}^{j'k'} \omega_{2^{m+1}}^{j'_0 k'_0} \omega_{2^m}^{j_0 k'_1} \dots \omega_4^{j_0 k'_{m-1}} |k'_0 k'_1 \dots k'_{m-1}\rangle |j_0\rangle$$

Using the fact that $\omega_N = \omega_{rN}^r$ for any choice of positive integers N and r , we may simplify the above expression and conclude that the state of the circuit after the

controlled phase shifts is

$$\begin{aligned} & \frac{1}{\sqrt{2^m}} \sum_{k'=0}^{2^m-1} \omega_{2^m+1}^{2j'k'+j'_0k'_0+j_0(2k'_1)+\dots+j_0(2^{m-1}k'_{m-1})} |k'_0k'_1\dots k'_{m-1}\rangle |j_0\rangle \\ &= \frac{1}{\sqrt{2^{m+1}}} \sum_{k'=0}^{2^m-1} \omega_{2^m+1}^{2^m-1} \omega_{2^m+1}^{jk'} |k'_0k'_1\dots k'_{m-1}\rangle |j_0\rangle \end{aligned}$$

Finally, the Hadamard transform maps this state to

$$\frac{1}{\sqrt{2^m+1}} \sum_{k'=0}^{2^m-1} \sum_{k_m=0}^1 (-1)^{k_m j_0} |k'_0k'_1\dots k'_{m-1}\rangle |k_m\rangle$$

Notice that

$$(-1)^{k_m j_0} = (-1)^{k_m j} = \omega_{2^m+1}^{j(2^m k_m)}$$

which implies that the final state is

$$\frac{1}{\sqrt{2^m+1}} \sum_{k'=0}^{2^m-1} \sum_{k_m=0}^1 \omega_{2^m+1}^{jk'+j(2^m k_m)} |k'_0k'_1\dots k'_{m-1}\rangle |k_m\rangle = \frac{1}{\sqrt{2^{m+1}+1}} \sum_{k=0}^{2^{m+1}-1} \omega_{2^m+1}^{jk} |k_0k_1\dots k_m\rangle$$

as required.

How many gates are required in the above circuit? Letting $g(m)$ denote the number of gates needed to perform \tilde{QFT}_{2^m} , we have the following recurrence:

$$\begin{aligned} g(1) &= 1 \\ g(m+1) &= g(m) + g(m+1) \end{aligned}$$

The solution to this recurrence is

$$g(m) = \sum_{j=1}^m j =^{m+1} C_2$$

Thus, we need only $O(m^2)$ gates to compute the quantum Fourier transform on m qubits. In fact there are better bounds known that are based on fast multiplication methods. However, these constructions are much more complicated and would probably not be practical (assuming we had a quantum computer) until m is quite large.

13.6 Summary

For any j in the computational basis $N = 2^n$ Fourier transform is

$$U_{FT} |j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{\iota 2\pi \frac{kj}{N}} |k\rangle$$

for $j = 0, \dots, N - 1$. Also,

$$U_{FT} |0^n\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} |k\rangle = H^{\otimes n} |0^n\rangle$$

Inverse Fourier transform is then defined by taking its conjugate as:

$$U_{FT}^\dagger |j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{-\iota 2\pi \frac{kj}{N}} |k\rangle$$

Note that $U_{FT}^T = U_{FT}$. Thus, $U_{FT}^\dagger = (U_{FT}^T)^* = U_{FT}^*$.

Let us now represent k, j in binary representation. Then

$$k = (k_{n-1} \dots k_0) \quad j = (j_{n-1} \dots j_0)$$

Thus,

$$k = k_{n-1}2^{n-1} + \dots + k_0; \quad j = j_{n-1}2^{n-1} + \dots + j_0$$

We have

$$\begin{aligned} \frac{kj}{2^n} &= \left(\frac{k_{n-1}}{2} + \dots + \frac{k_0}{2^n} \right) j \\ &= \frac{k_{n-1}j}{2} + \dots + \frac{k_0j}{2^n} \\ &= k_{n-1} \left(j_{n-1}2^{n-2} + \dots + \frac{j_0}{2} \right) + \dots + k_0 \left(\frac{j_{n-1}}{2} + \dots + \frac{j_0}{2^n} \right) \\ &= k_{n-1}(j_{n-1} \dots j_1 \cdot j_0) + \dots + k_0(0 \cdot j_{n-1} \dots j_0) \\ \frac{kj}{2^n} &= k_0(0 \cdot j_{n-1} \dots j_0) + \dots + k_{n-1}(j_{n-1} \dots j_1 \cdot j_0) \end{aligned}$$

Thus, substituting in the exponential $e^{\iota 2\pi \frac{kj}{2^n}}$ can be written as

$$\begin{aligned} e^{\iota 2\pi \frac{kj}{2^n}} &= e^{\iota 2\pi k_0(0 \cdot j_{n-1} \dots j_0) + \dots + k_{n-1}(j_{n-1} \dots j_1 \cdot j_0)} \\ &= e^{\iota 2\pi k_0(0 \cdot j_{n-1} \dots j_0)} \cdot \dots \cdot e^{\iota 2\pi k_{n-1}(j_{n-1} \dots j_1 \cdot j_0)} \end{aligned}$$

Thus, now using the fact that $2^{\iota 2\pi k(p,q)} = e^{\iota 2\pi k(p+0,q)} = e^{\iota 2\pi kp} 2^{\iota 2\pi k0.q} = e^{\iota 2\pi k(0.q)}$ for integer p , as $e^{2\pi i kp} = 1$.

$$\begin{aligned}
 U_{FT} |j_{n-1} \dots j_0\rangle &= \frac{1}{\sqrt{2^n}} \sum_{k_{n-1}, \dots, k_0} (e^{\iota 2\pi(k_0(0.j_{n-1} \dots j_0))} \dots e^{\iota 2\pi k_{n-1}(j_{n-1} \dots j_1.j_0)}) |k_{n-1} \dots k_0\rangle \\
 &= \frac{1}{\sqrt{2^n}} \sum_{k_{n-1}, \dots, k_0} (e^{\iota 2\pi k_{n-1}(0.j_0)} \dots e^{\iota 2\pi(k_0(0.j_{n-1} \dots j_0))}) |k_{n-1} \dots k_0\rangle \\
 &= \frac{1}{\sqrt{2^n}} \sum_{k_{n-1}, \dots, k_0} (e^{\iota 2\pi k_{n-1}(0.j_0)} |k_{n-1}\rangle \otimes \dots \otimes e^{\iota 2\pi(k_0(0.j_{n-1} \dots j_0))} |k_0\rangle) \\
 &= \frac{1}{\sqrt{2^n}} \sum_{k_{n-1}, \dots, k_0} \otimes_{l=n-1}^0 e^{\iota 2\pi k_l(0.j_{l-n+1} \dots j_0)} |k_l\rangle \\
 &= \frac{1}{\sqrt{2^n}} \left(\sum_{k_{n-1}=0}^1 e^{\iota 2\pi k_{n-1}(0.j_0)} |k_{n-1}\rangle \right) \otimes \left(\sum_{k_{n-2}=0}^1 e^{\iota 2\pi k_{n-2}(0.j_1.j_0)} |k_{n-2}\rangle \right) \otimes \dots \otimes \\
 &\quad \left(\sum_{k_0=0}^1 e^{\iota 2\pi k_0(0.j_{n-1} \dots j_0)} |k_0\rangle \right) \\
 &= \frac{1}{\sqrt{2^n}} \otimes_{l=n-1}^0 \left(\sum_{k_l=0}^1 e^{\iota 2\pi k_l(0.j_{l-n+1} \dots j_0)} |k_l\rangle \right) \\
 \implies U_{FT} |j_{n-1} \dots j_0\rangle &= \left(\frac{|0\rangle + e^{\iota 2\pi(0.j_0)|1\rangle}}{\sqrt{2}} \right) \otimes \left(\frac{|0\rangle + e^{\iota 2\pi(0.j_1.j_0)|1\rangle}}{\sqrt{2}} \right) \otimes \dots \otimes \\
 &\quad \left(\frac{|0\rangle + e^{\iota 2\pi(0.j_{n-1} \dots j_0)|1\rangle}}{\sqrt{2}} \right)
 \end{aligned}$$

Thus, we need to implement manipulations of the form

$$|j_0\rangle \rightarrow \frac{1}{\sqrt{2}} (|0\rangle + e^{\iota 2\pi(0.j_{n-1} \dots j_0)} |1\rangle)$$

using controlled rotations. Note that,

$$e^{\iota 2\pi(0.j_{n-1} \dots j_0)} = e^{\iota 2\pi(0.j_{n-1})} e^{\iota 2\pi(0.0.j_{n-2})} \dots e^{\iota 2\pi(0.0 \dots 0.j_0)}$$

which can be done by implementing controlled rotations in sequence. Recall the Phase gate

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

Putting $R_l = P(\pi/2^{l-1})$. In particular, $R_1 = Z$. Thus, we get,

$$P(\pi/2^{l-1}) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/2^{l-1}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^l} \end{bmatrix} = R_l$$

Hence, clearly, $R_l |0\rangle = |0\rangle$ and $R_l |1\rangle = e^{i\pi/2^{l-1}} |1\rangle$.

Implementation of controlled rotations

Consider the implementation of

$$|j\rangle \rightarrow \frac{1}{\sqrt{2}} (|0\rangle + e^{i2\pi(0.j_{n-1}\dots j_0)} |1\rangle)$$

where $|j\rangle = |j_{n-1}\dots j_0\rangle$. When $n = 1$, we need to implement

$$|j_0\rangle \rightarrow \left(\frac{|0\rangle + e^{i2\pi(0.j_0)} |1\rangle}{\sqrt{2}} \right)$$

This is the Hadamard gate:

$$|j_0\rangle \rightarrow H |j_0\rangle$$

When $n = 2$, we need to implement

$$|j_1 j_0\rangle \rightarrow \frac{1}{\sqrt{2^2}} (|0\rangle + e^{i2\pi(0.j_0)} |1\rangle) \otimes (|0\rangle + e^{i2\pi(0.j_1 j_0)} |1\rangle)$$

Now, consider the implementation of the following circuit in figure 13.11. **Analysis:**

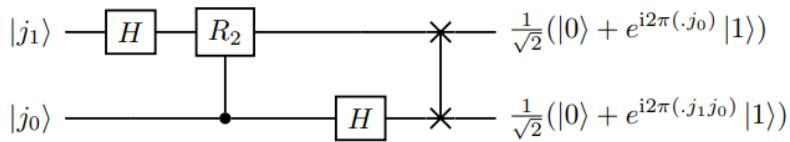


Figure 13.11: QFT_2

$$|j_1 j_0\rangle \xrightarrow{H \otimes I} \frac{1}{\sqrt{2}} (|0\rangle + e^{i2\pi(0.j_1)} |1\rangle) |j_0\rangle$$

We then apply the controlled R_2 (with $|j_0\rangle$ as the control bit). Now, $R_2 = P(\pi/2) = P(2\pi/4)$,

$$\begin{aligned} &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & e^{i2\pi/4} \end{pmatrix} \begin{pmatrix} 1 \\ e^{i2\pi(0.j_1)} \end{pmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ e^{i2\pi(0.j_1)} \end{pmatrix} \\ \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ e^{i2\pi(0.j_1 j_0)} \end{pmatrix} &= \begin{cases} \frac{1}{\sqrt{2}}(|0\rangle + e^{i2\pi(0.j_1)}|1\rangle) & \text{if } j_0 = 1 \\ \frac{1}{\sqrt{2}}(|0\rangle + e^{i2\pi(0.j_1 0)}|1\rangle) & \text{if } j_0 = 0 \end{cases} \\ \implies \frac{1}{\sqrt{2}}(|0\rangle + e^{i2\pi(0.j_1 j_0)}|1\rangle) \otimes |j_0\rangle & \end{aligned}$$

Now applying the Hadamard gate on $|j_0\rangle$ and the finally swapping the qubits, we get,

$$\begin{aligned} &\xrightarrow{I \otimes H} \left(\frac{1}{\sqrt{2}}(|0\rangle + e^{i2\pi(0.j_1 j_0)}|1\rangle) \right) \otimes \left(\frac{1}{\sqrt{2}}(|0\rangle + e^{i2\pi(0.j_0)}|1\rangle) \right) \\ &\xrightarrow{\text{SWAP}} \left(\frac{1}{\sqrt{2}}(|0\rangle + e^{i2\pi(0.j_0)}|1\rangle) \right) \otimes \left(\frac{1}{\sqrt{2}}(|0\rangle + e^{i2\pi(0.j_1 j_0)}|1\rangle) \right) \end{aligned}$$

Thus, the required result. In general this can be extended to n qubits as shown through the following quantum circuit. and then finally swapping, we can get the

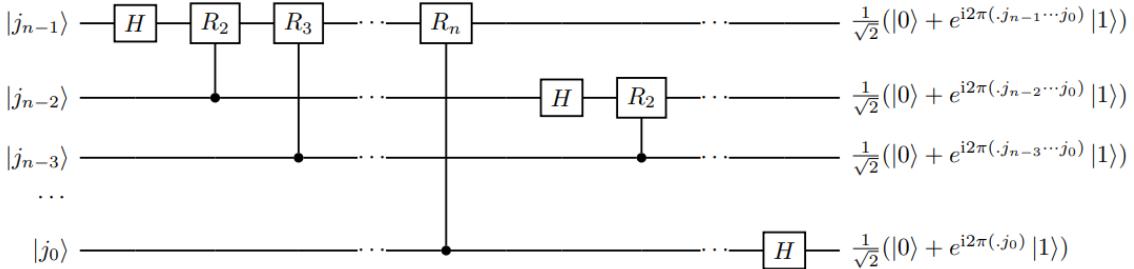
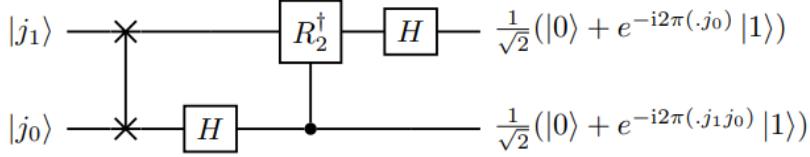


Figure 13.12: QFT_N

required result. Similarly, the circuit for inverse Fourier transform can also be implemented. For example, for the case $n = 2$, the circuit is as shown in the figure 13.13. In general, the DFT can be thought of as follows:

Figure 13.13: QFT_2^\dagger

$$\frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_N & \omega_N^2 & \dots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \dots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \dots & \omega_N^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} = \sum_{k=0}^{N-1} \sum_{j=0}^{N-1} x_j e^{2\pi i j k / N} |k\rangle$$

where $\omega_N = e^{2\pi i / N}$.

Example 13.6.1. (Classical Fast Fourier transform) Suppose we wish to perform a Fourier transform of a vector containing 2^n complex numbers on a classical computer. Verify that the straightforward method for performing the Fourier transform based upon direct evaluation of equation

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k / N}$$

requires $\Theta(2^{2n})$ elementary arithmetic operations. Find a method for reducing this to $\Theta(n2^n)$ operations, based upon equation

$$|j_{n-1} \dots j_0\rangle \rightarrow \frac{(|0\rangle + e^{2\pi i 0 \cdot j_0}|1\rangle)(|0\rangle + e^{2\pi i 0 \cdot j_1 j_0}|1\rangle) \dots (|0\rangle + e^{2\pi i 0 \cdot j_{n-1} \dots j_0}|1\rangle)}{2^{n/2}}$$

Note that for evaluating each y_k we are required to perform $N - 1$ additions and N multiplications, thus $2 \times 2^n - 1$ additions. Since there are 2^n y'_k s thus the total complexity would be $\Theta(2^{2n})$.

(Cooley–Tukey Algorithm) For each x_k we can separate the sum into odd and even indices, then we require 2^n operations assuming the two separate sums are known. This can be done recursively, splitting each sum into 2 pieces. This leads to the number of operations to be $\Theta(2^n \log_2 2^n) = \Theta(n2^n)$.

A radix-2 decimation-in-time (DIT) FFT is the simplest and most common form of the Cooley–Tukey algorithm, although highly optimized Cooley–Tukey implementations typically use other forms of the algorithm as described below. Radix-2 DIT

divides a DFT of size N into two interleaved DFTs (hence the name "radix-2") of size $N/2$ with each recursive stage.

The discrete Fourier transform (DFT) is defined by the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk},$$

where k is an integer ranging from 0 to $N - 1$.

Radix-2 DIT first computes the DFTs of the even-indexed inputs

$$(x_{2m} = x_0, x_2, \dots, x_{N-2})$$

and of the odd-indexed inputs

$$(x_{2m+1} = x_1, x_3, \dots, x_{N-1}),$$

and then combines those two results to produce the DFT of the whole sequence. This idea can then be performed recursively to reduce the overall runtime to $O(N \log N)$. This simplified form assumes that N is a power of two; since the number of sample points N can usually be chosen freely by the application (e.g., by changing the sample rate or window, zero-padding, etc.), this is often not an important restriction.

The radix-2 DIT algorithm rearranges the DFT of the function x_n into two parts: a sum over the even-numbered indices $n = 2m$ and a sum over the odd-numbered indices $n = 2m + 1$:

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N} (2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N} (2m+1)k}$$

One can factor a common multiplier $e^{-\frac{2\pi i}{N} k}$ out of the second sum, as shown in the equation below. It is then clear that the two sums are the DFT of the even-indexed part x_{2m} and the DFT of odd-indexed part x_{2m+1} of the function x_n . Denote the DFT of the even-indexed inputs x_{2m} by E_k and the DFT of the odd-indexed inputs x_{2m+1} by O_k , and we obtain:

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2} mk} + e^{-\frac{2\pi i}{N} k} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2} mk} = E_k + e^{-\frac{2\pi i}{N} k} O_k$$

for $k = 0, \dots, N/2 - 1$.

Note that the equalities hold for $k = 0, \dots, N - 1$, but the crux is that E_k and O_k are calculated in this way for $k = 0, \dots, N/2 - 1$ only. Thanks to the periodicity of the complex exponential, $X_{k+N/2}$ is also obtained from E_k and O_k :

$$X_{k+N/2} = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2} m(k+N/2)} + e^{-\frac{2\pi i}{N} (k+N/2)} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2} m(k+N/2)}$$

Expanding and simplifying:

$$\begin{aligned} X_{k+N/2} &= \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2} mk} e^{-2\pi mi} + e^{-\frac{2\pi i}{N} k} e^{-\pi i} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2} mk} e^{-2\pi mi} \\ &= \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2} mk} - e^{-\frac{2\pi i}{N} k} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2} mk} \\ &= E_k - e^{-\frac{2\pi i}{N} k} O_k \end{aligned}$$

We can rewrite X_k and $X_{k+N/2}$ as:

$$X_k = E_k + e^{-\frac{2\pi i}{N} k} O_k, \quad X_{k+N/2} = E_k - e^{-\frac{2\pi i}{N} k} O_k$$

This result, expressing the DFT of length N recursively in terms of two DFTs of size $N/2$, is the core of the radix-2 DIT fast Fourier transform. The algorithm gains its speed by re-using the results of intermediate computations to compute multiple DFT outputs. Note that final outputs are obtained by a \pm combination of E_k and $O_k e^{-\frac{2\pi i}{N} k}$, which is simply a size-2 DFT (sometimes called a butterfly in this context); when this is generalized to larger radices below, the size-2 DFT is replaced by a larger DFT (which itself can be evaluated with an FFT).

Example 13.6.2. Give a decomposition of the controlled R_l gate into single qubit and $CNOT$ gates.

Let $R_k = e^{i\alpha} AXBXC$ with $ABC = I$. Taking $\alpha = \frac{\pi}{2^k}$, $A = I$, $B = R_Z(-\frac{\pi}{2^k})$ and $C = R_Z(\frac{\pi}{2^k})$ we see that $ABC = I$ and

$$\begin{aligned} AXBXC &= XR_Z(-\frac{\pi}{2^k})XR_Z(\frac{\pi}{2^l}) \\ &= R_Z(\frac{\pi}{2^k})R_z(\frac{\pi}{2^k}) \\ &= R_Z\left(\frac{2\pi}{2^k}\right) \end{aligned}$$

Thus, $R_k = e^{\iota\pi/2^k} R_Z \left(\frac{\pi\iota}{2^k} \right)$ which is:

$$R_k = e^{\iota\pi/2^k} \begin{bmatrix} e^{-\iota\pi/2^k} & 0 \\ 0 & e^{\iota\pi/2^k} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi\iota/2^k} \end{bmatrix}$$

Thus, the following circuit in figure 13.14:

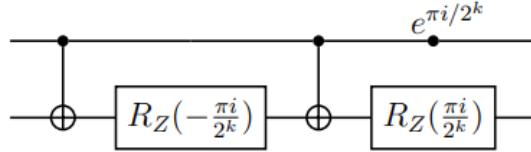


Figure 13.14: Implementation of controlled phase gate

Example 13.6.3. (Approximate quantum Fourier transform) The quantum circuit construction of the quantum Fourier transform apparently requires gates of exponential precision in the number of qubits used. However, such precision is never required in any quantum circuit of polynomial size. For example, let U be the ideal quantum Fourier transform on n qubits, and V be the transform which results if the controlled R_l gates are performed to a precision $\Delta = 1/p(n)$ for some polynomial $p(n)$. Show that the error $E(U, V) \equiv \max_{|\psi\rangle} \|(U - V)|\psi\rangle\|$ scales as $\Theta(n^2/p(n))$, and thus polynomial precision in each gate is sufficient to guarantee polynomial accuracy in the output state.

In the circuit we have $m = \frac{n(n+1)}{2} = \Theta(n^2)$ R_k gates. Using the results, $E(U, V) = \max_{|\psi\rangle} \|(U - V)|\psi\rangle\| \leq m \frac{1}{p(n)} = \Theta(\frac{n^2}{p(n)})$. Thus, the error scales as $\Theta(n^2/p(n))$ and thus polynomial precision in each gate is sufficient to guarantee polynomial accuracy in the output state.

13.7 Analysis

13.7.1 Total Gate Complexity

From the figure, we can clearly see that the total number of gates are $n + n - 1 + \dots + 1 = \frac{n(n-1)}{2}$ and including the $n/2$ swap $\mathcal{O}(n/2)$ swap operations, we get,

$$\frac{n(n-1)}{2} + \frac{n}{2} = \frac{n^2}{2} = \mathcal{O}(n^2)$$

Thus, $\mathcal{O}(n^2)$ gate complexity in total.

The construction itself proves that the quantum Fourier Transform is Unitary, since each gate in the circuit is unitary. Each circuit uses for the first qubit 1 Hadamard gate and $n - 1$ conditional gates. This is followed by a Hadamard gate and $n - 2$ conditional rotations on the second qubit, for a total of $n + (n - 1)$ gates. Continuing this way, we see that $n + (n - 1) + \dots + 1 = n(n + 1)/2$ gates are required, plus the gates involved in the swaps. At most $n/2$ swaps are required, and each swap can be accomplished using three controlled $C - NOT$ gates. Therefore, this circuit provides a $\Theta(n^2)$ algorithm for performing the quantum Fourier Transform.

13.7.2 Comparison with Discrete Fourier Transform

In contrast, the best classical algorithm for computing the discrete Fourier transform on 2^n elements are algorithms such as the Fast Fourier Transform (FFT), which compute the discrete Fourier transform using $\Theta(n2^n)$ gates. That is, it requires exponentially more operations to compute the Fourier transform on a quantum computer.

At face value this sounds terrific, since the Fourier transform is a crucial step in so many real-world data processing applications. For example, in computer speech recognition, the first step in phoneme recognition is to Fourier transform the digitized sound. Can we use the quantum Fourier transform to speed up the computation of these Fourier transforms? Unfortunately, the answer is that there is no known way to do this. The problem is that the amplitudes in a quantum computer cannot be directly accessed by measurement. Thus, there is no way of determining the Fourier transformed amplitudes of the original state. Worse still, there is in general no way to efficiently prepare the original state to be Fourier transformed. Thus, finding uses for the quantum Fourier transform is more subtle than we might have hoped.

13.8 Implementation using Qiskit

In this section we implement the QFT in using the inbuilt Qiskit circuit library implementation and then built the QFT from scratch and show that both the implementation are the same. Same as is done with Inverse Quantum Fourier Transform.

13.8.1 Quantum Fourier Transform

We will now see the implementation of QFT in Qiskit. For this, we will first draw the QFT circuit using the inbuilt qiskit circuit library.

```

1 # Importing Libraries
2
3 from qiskit import QuantumCircuit
4 import qiskit.quantum_info as qi
5 from qiskit.circuit.library import QFT
6 from qiskit import transpile
7 from qiskit_aer import AerSimulator
8
9 simulator=AerSimulator()

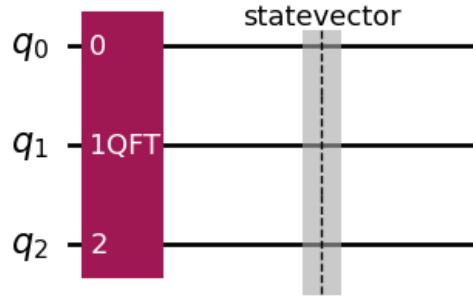
```

We are going to use Aer simulator in order to simulate and compare our implementation with the inbuilt Qiskit implementation

```

1 n=3
2 qc_QFT = QuantumCircuit(n)
3 qc_QFT.append(QFT(n), range(n))
4 qc_QFT.save_statevector()
5 qc_QFT.draw('mpl')

```



```

1 results_QFT = simulator.run(transpile(qc_QFT, simulator)).result()
2 psi=results_QFT.get_statevector()
3 psi.draw('latex', prefix='{\ket{\psi} = } ')

```

This results in the following state vector:

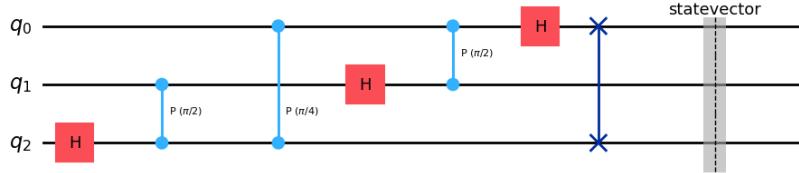
$$|\psi\rangle = \frac{\sqrt{2}}{4}|000\rangle + \frac{\sqrt{2}}{4}|001\rangle + \dots + \frac{\sqrt{2}}{4}|111\rangle$$

Thus, we have shown the implementation of QFT using inbuilt Qiskit circuit library. Now, we will do the same by writing a function for building the QFT for n qubit. Note that because Qiskit follows Little Endian notation, the qubits are reversed. So the qubit 0 is the last one on the list. Thus circuit is implemented as in the following code.

```

1 def QFT_custom(n):
2     qc_QFT_custom = QuantumCircuit(n, name='QFT')
3     for i in range(n-1,-1,-1):
4         qc_QFT_custom.h(i)
5         for j in range(i-1,-1,-1):
6             qc_QFT_custom.cp(np.pi/2**((i-j)),j,i)
7
8     for i in range(n//2):
9         qc_QFT_custom.swap(i,n-i-1)
10
11    return qc_QFT_custom
12
13 #qc_QFT_custom = QuantumCircuit(n)
14 #qc_QFT_custom.append(QFT_custom(n), range(n))
15 qc_QFT_custom = QFT_custom(n)
16 qc_QFT_custom.save_statevector()
17 # save it in images folder
18 qc_QFT_custom.draw('mpl', filename='../images/qft_custom.png')

```



```

1 qc_transpiled= transpile(qc_QFT_custom, simulator)
2 result = simulator.run(transpile(qc_QFT_custom, simulator)).result()
3 psi_custom = result.get_statevector()
4 psi_custom.draw('latex', prefix='{\ket{\psi} = }')

```

This also outputs the same state:

$$\frac{\sqrt{2}}{4} |000\rangle + \frac{\sqrt{2}}{4} |001\rangle + \dots + \frac{\sqrt{2}}{4} |111\rangle$$

Thus, the explicit circuit for the three qubit Quantum Fourier transform is as shown in the figure. Note that we have used Phase gates for implementation of the controlled rotations. As a matrix the quantum Fourier transform in this instance may be written

out explicitly, using $\omega = e^{2\pi i/8} = \sqrt{i}$, as

$$\frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega^1 & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^3 & \omega^2 & \omega^1 \end{bmatrix}$$

This can also be explicitly seen using the following code

```
1 QFT_matrix = qi.Operator(QFT(n))
2 print(QFT_matrix.data)
```

which outputs the same matrix. Thus, the implemented circuit as well as the qiskit circuit library implementation all are the same.

13.8.2 Inverse Quantum Fourier Transform

We can similarly also implement the inverse Fourier transform using qiskit circuit library by just adding the inverse method to the call as shown in the following code

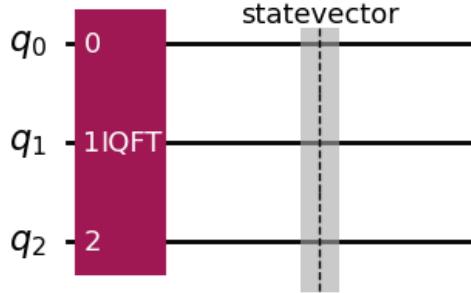
```
1 def QFT_dagger_circuit(n):
2     qc_QFT_dagger = QuantumCircuit(n)
3     qc_QFT_dagger.append(QFT(n).inverse(), range(n))
4     return qc_QFT_dagger
5
6 qc_QFT_dagger = QFT_dagger_circuit(n)
7 qc_QFT_dagger.save_statevector()
8 qc_QFT_dagger.draw('mpl')
```

The code for output of the state vector on the $|000\rangle$ as input from the computational basis can be found out using the following code as:

```
1 res = simulator.run(transpile(qc_QFT_dagger, simulator)).result()
2 psi_dagger = res.get_statevector()
3 psi_dagger.draw('latex', prefix='{\backslash ket{\backslash psi} = }')
```

The custom implementation for n qubit of inverse Fourier Transform is as shown in the following code

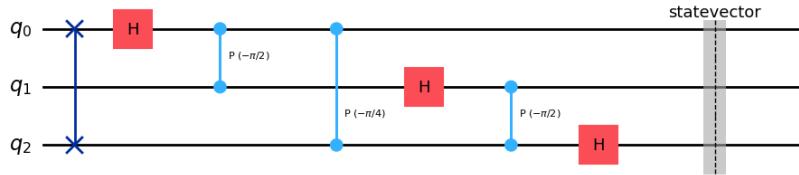
```
1 def IQFT_custom(n):
2     qc_IQFT_custom = QuantumCircuit(n, name='IQFT')
3     qc_IQFT_custom.x(1)
```



```

4     for i in range(n//2):
5         qc_IQFT_custom.swap(i, n-i-1)
6
7     for i in range(n):
8         qc_IQFT_custom.h(i)
9         for j in range(i+1,n):
10            qc_IQFT_custom.cp(-np.pi/2**((j-i)), j, i)
11
12
13
14     return qc_IQFT_custom
15
16 qc_IQFT_custom = IQFT_custom(n)
17
18 qc_IQFT_custom.save_statevector()
19 qc_IQFT_custom.draw('mpl', filename='../images/iqft_custom.png')

```



```

1 result = simulator.run(transpile(qc_IQFT_custom, simulator)).result()
2 psi_iqft_custom = result.get_statevector()
3 psi_iqft_custom.draw('latex', prefix='{\\ket{\psi} = }')

```

The state vector is the same in both cases as

$$\frac{\sqrt{2}}{4} |000\rangle + \dots + \frac{\sqrt{2}}{4} |111\rangle$$

Using the following code, we can show that the unitary matrix is same in both the cases. Thus, our implementation is correct.

```
1 IQFT_matrix = qi.Operator(QFT(n).inverse())
2 print(IQFT_matrix.data)
3 IQFT_custom_matrix = qi.Operator(IQFT_custom(n))
4 print(IQFT_custom_matrix.data)
```

Chapter 14

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 $x \in \{0, 1\}^n$.

14.1 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 $2^n \times 2^n$. If n is reasonably large such as $n = 1000$, it would be impossible to write down an explicit description of U because it is too large - all of the computers in the world could only store a fraction of its entries. Even if you just wanted to compute a single entry of U from the description of Q , you might be faced with a computationally difficult task. 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 θ 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 $j = 1, 2, \dots, 2^n$. The reason why the eigenvalues has the form $e^{2\pi i\theta}$, which is equivalent to saying that these eigenvalues are on the complex unit circle, is that U is unitary and therefore preserves Euclidean length. Thus, now we define the **Phase Estimation Problem** as follows:

14.2 The Problem Definition

Problem 14.2.1. *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 $\theta \in [0, 1)$.

The Phase Estimation has applications in Shor's Algorithm, solving system of Linear Equations (HHL), Eigenvalue problems, Amplitude estimations, Quantum Counting and Quantum walks.

Remark. Note that we have made no specific requirements on the precision to which θ must be approximated. It will turn out that for an arbitrary Quantum circuit Q

and eigen vector $|\psi\rangle$, the number θ 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.

14.3 The Classical Solution

Using a classical computer, we can estimate θ 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.

14.4 The Quantum Solution

Before going into solving the Problem, we will look a some of the pre-requisites for attacking the problem efficiently.

14.4.1 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 14.1. Note that the circuit transforms $|0\rangle|\psi\rangle$ as:

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

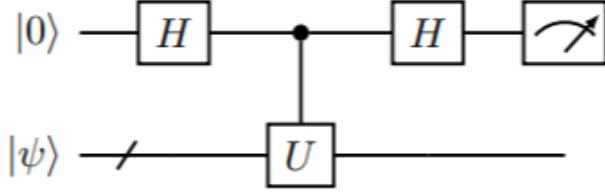


Figure 14.1: Hadamard Test

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 $\sqrt{\langle\psi|(I + U^\dagger)(I + U)|\psi\rangle}$ and upon solving for this we get the following:

$$\begin{aligned} &= \sqrt{\langle\psi|(I + U^\dagger)(I + U)|\psi\rangle} \\ &= \sqrt{\langle\psi|I + U + U^\dagger + U^\dagger U|\psi\rangle} \\ &= \sqrt{\langle\psi|2I + U + U^\dagger|\psi\rangle} \\ &= \sqrt{2\langle\psi|\psi\rangle + \langle\psi|U|\psi\rangle + \langle\psi|U^\dagger|\psi\rangle} \\ &= \sqrt{2 + \langle\psi|U|\psi\rangle + (\langle\psi|U|\psi\rangle)^\dagger} \\ &= \sqrt{2 + 2\text{Re}(\langle\psi|U|\psi\rangle)} \quad \text{Using the fact that } a + a^\dagger = 2\text{Re}(a); a = \langle\psi|U|\psi\rangle \in \mathbb{C} \end{aligned}$$

Similarly, $\sqrt{\langle\psi|(I-U)(I-U^\dagger)|\psi\rangle} = \sqrt{2 - 2Re(\langle\psi|U|\psi\rangle)}$. Now, substituting this into the equation, we get,

$$\frac{\sqrt{2 + 2Re(\langle\psi|U|\psi\rangle)}}{2} \left(|0\rangle \otimes \frac{(|\psi\rangle + U|\psi\rangle)}{\sqrt{2 + 2Re(\langle\psi|U|\psi\rangle)}} \right) + \\ \frac{\sqrt{2 - 2Re(\langle\psi|U|\psi\rangle)}}{2} \left(|1\rangle \otimes \frac{(|\psi\rangle - U|\psi\rangle)}{\sqrt{2 - 2Re(\langle\psi|U|\psi\rangle)}} \right)$$

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 $-1 \leq Re(\langle\psi|U|\psi\rangle) \leq 1$. Thus, we can perform several runs of the circuit to get an approximate $p(0)$ to get an approximate $p(0)$ and thus can estimate $Re(\langle\psi|U|\psi\rangle)$ using

$$Re(\langle\psi|U|\psi\rangle) = 2p(0) - 1$$

To obtain the imaginary part we can use the circuit as shown in the figure 14.2 called the (imaginary) Hadamard test. Similarly doing similar calculations as done

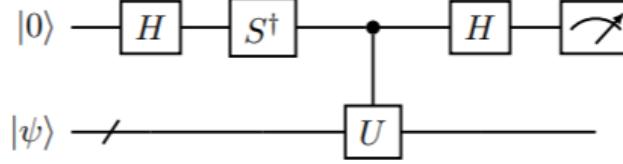


Figure 14.2: Hadamard Test

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:

$$p(0) = \frac{1}{2}(1 + Im(\langle\psi|U|\psi\rangle))$$

Thus, we can perform several runs on the circuit to get an approximate $p(0)$ to estimate $Im(\langle\psi|U|\psi\rangle)$.

$$Im(\langle\psi|U|\psi\rangle) = 2p(0) - 1$$

Thus, combining the results of the two circuits, we obtain the estimate to $\langle \psi | U | \psi \rangle$. Thus, the idea is to run both the real and imaginary Hadamard Test circuits (can be done in parallel) multiple times in order to approximate $p(0)$ from both (for real and imaginary parts) and then subsequently calculate

$$\langle \psi | U | \psi \rangle = \text{Re}(\langle \psi | U | \psi \rangle) + i\text{Im}(\langle \psi | U | \psi \rangle)$$

14.4.2 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 14.3. Note that it is exactly the Hadamard test with U

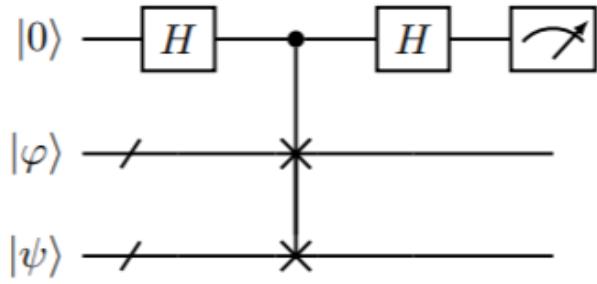


Figure 14.3: Swap Test

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:

$$\frac{1}{2}(|0\rangle|\phi\rangle|\psi\rangle + |0\rangle|\psi\rangle|\phi\rangle - |1\rangle|\psi\rangle|\phi\rangle + |1\rangle|\phi\rangle|\psi\rangle)$$

$$= \frac{\sqrt{2+2|\langle\psi|\phi\rangle|^2}}{2} \left(|0\rangle \otimes \frac{|\phi\rangle|\psi\rangle + |\psi\rangle|\phi\rangle}{\sqrt{2+2|\langle\psi|\phi\rangle|^2}} \right) + \frac{\sqrt{2-2|\langle\psi|\phi\rangle|^2}}{2} \left(|1\rangle \otimes \frac{|\phi\rangle|\psi\rangle - |\psi\rangle|\phi\rangle}{\sqrt{2-2|\langle\psi|\phi\rangle|^2}} \right)$$

Now, upon measurement of the first qubit, the probability of measuring the first qubit to be 0 is:

$$p(0) = \frac{1}{2}(1 + |\langle\phi|\psi\rangle|^2)$$

Thus using multiple runs of this circuit we can approximate $p(0)$ and thus can estimate $|\langle\psi|\phi\rangle|^2$ overlap using

$$|\langle\psi|\phi\rangle|^2 = 2p(0) - 1$$

14.4.3 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 $U = U_\phi^\dagger U_\psi$ in the (real) and (imaginary) Hadamard Test and putting the state $|\psi\rangle = |0^n\rangle$ in the second register, we get, $\langle\phi|\psi\rangle = \langle 0^n | U | 0^n \rangle$, thus calculating the expectation value, we can obtain the real and imaginary parts of $\langle\phi|\psi\rangle$ after getting approximate $p(0)$ real and hadmard tests and thus getting $Re(\langle 0^n | U_{phi}^\dagger U_\psi | 0^n \rangle)$ and $Im(\langle 0^n | U_{phi}^\dagger U_\psi | 0^n \rangle)$ respectively using

$$Re(\langle 0^n | U_\phi^\dagger U_\psi | 0^n \rangle) = 2p(0) - 1$$

$$Im(\langle 0^n | U_\phi^\dagger U_\psi | 0^n \rangle) = 2p(0) - 1$$

Recall that e get each $p(0)$ from different circuits i.e. $p(0)$ are not necessarily same in both the cases. Thus, we get

$$\langle 0^n | U_\phi^\dagger U_\psi | 0^n \rangle = Re(\langle 0^n | U_\phi^\dagger U_\psi | 0^n \rangle) + \iota Im(\langle 0^n | U_\phi^\dagger U_\psi | 0^n \rangle)$$

$$\langle\psi|\phi\rangle = Re(\langle\psi|\phi\rangle) + \iota Im(\langle\psi|\phi\rangle)$$

14.4.4 Single Qubit 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 14.1. with U, ψ 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:

$$\begin{aligned} p(1) &= \frac{1}{2}(1 - Re(\langle\psi|U|\psi\rangle)) \\ &= \frac{1}{2}(1 - Re(\langle\psi|e^{2\pi i\theta}|\psi\rangle)) \\ &= \frac{1}{2}(1 - Re(e^{2\pi i\theta}\langle\psi|\psi\rangle)) \\ &= \frac{1}{2}(1 - Re(e^{2\pi i\theta})) \\ p(1) &= \frac{1}{2}(1 - \cos(2\pi\theta)) \end{aligned}$$

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 θ) to additive precision ϵ , the number of samples need is $\mathcal{O}(1/\epsilon^2)$ using the formula derived from statistics (If we know that the count X of "successes" in a group of n observations with success 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 θ is very close to 0 and we would like to estimate θ to additive precision ϵ . Note that

$$\begin{aligned} p(1) &= \frac{1}{2} \left(1 - \left(1 - \frac{(2\pi\theta)^2}{2} \right) \right) \\ p(1) &= \frac{(2\pi\theta)^2}{4} \\ p(1) &\approx (2\pi\theta)^2 = \mathcal{O}(\epsilon^2) \end{aligned}$$

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 θ is very close to 1/2 or 1 is similar.

Note that the circuit in figure 14.1 cannot distinguish the sign of θ (or whether $\theta \geq 1/2$ when restricted to the interval $[0, 1]$). In order to understand this consider the plot between $p(1)$ and θ as shown in the figure 14.4. For a value of $p(1)$, two

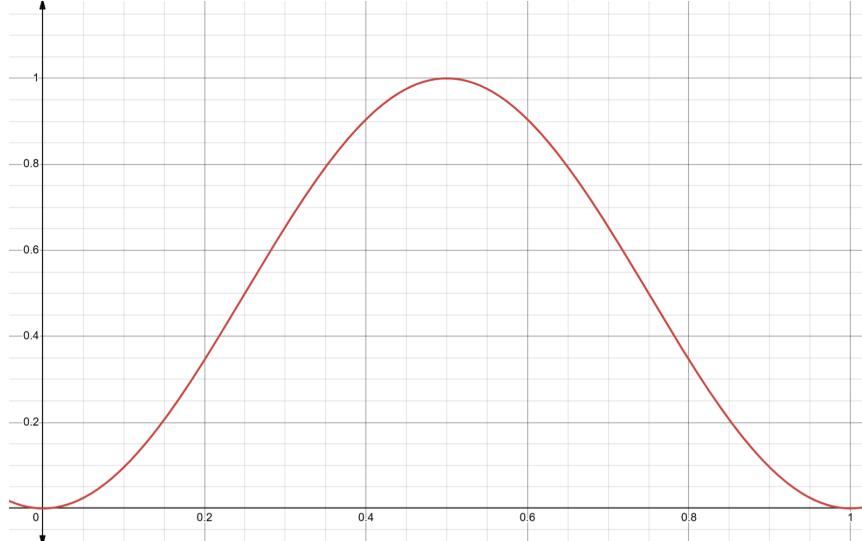


Figure 14.4: cosine plot

value of θ correspond to it. One value of $\theta > 1/2$ and one value of $\theta < 1/2$. Thus, upon running the circuit multiple times we get an approximate value of $p(1)$. But for a value of $p(1)$ there are two values of θ satisfying the equation. ($\theta > 1/2$ and $\phi < 1/2$). Thus, we cannot determine exactly which θ it is using Real Hadamard test. Hence, we use Imaginary Hadamard Test. To do this, we can use the circuit in figure 14.2 by replacing S^\dagger with S , so that the success probability of measuring 1 in

the computational basis is

$$p(1) = \frac{1}{2}(1 + \sin(2\pi\theta))$$

The corresponding graph is as shown in the figure 14.5. Thus, upon running the

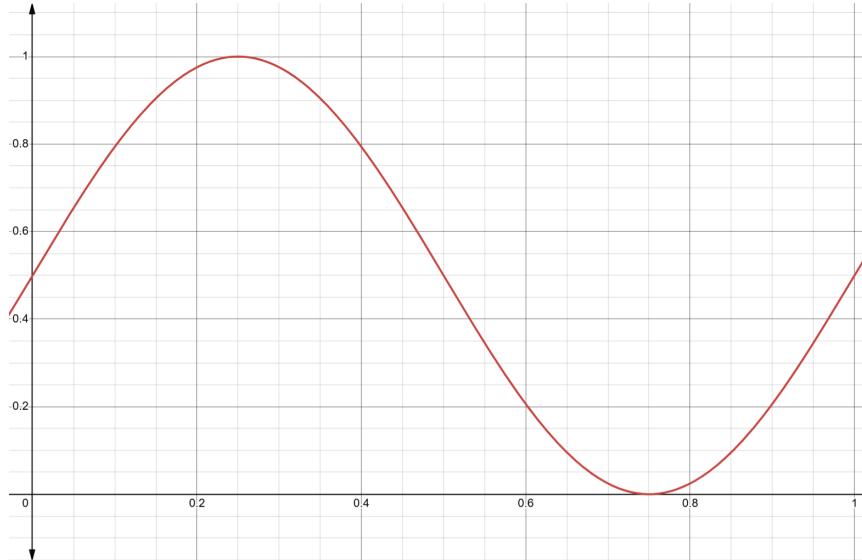


Figure 14.5: sine plot

circuit a few times we get an approximate value of $p(1)$. Now clearly from the figure we can see that if

$$\begin{cases} p(1) > 1/2 \implies \theta < 1/2 \\ p(1) < 1/2 \implies \theta > 1/2 \end{cases}$$

Note that we need run only sufficient times to get the first value i.e. $O(1)$ accuracy after the decimal (unless the value of $p(1)$ is close to 0.5) to find the range of ϕ . This gives

$$\theta = \begin{cases} \in [0, 1/2], & p(1) \geq \frac{1}{2} \\ \in (1/2, 1), & p(1) < \frac{1}{2} \end{cases}$$

Thus, using Imaginary hadamard Test we can find the range where ϕ lies (either $> 1/2$ or $< 1/2$) and then using the runs from real Hadamard test we can find approximate value of $p(1)$ and thus combining the two results we can find a good approximation of ϕ . Unlike previous estimate, in order to correctly estimate the sign,

we only require $\mathcal{O}(1)$ accuracy, and run figure 14.2 for a constant number of times (unless θ very close to 0 or π).

In conclusion for finding θ to an additive precision of ϵ , we need to find $p(1)$ to an additive precision of ϵ . In order to do so, we need to run the circuit (no. of samples) $O(\epsilon^2)$ times.

$$\begin{cases} \text{if } p = 0 \text{ or } p = 1 & \implies \text{Only one run is sufficient} \\ \text{if } p = 1/2 & \implies N \geq \frac{1}{4\epsilon^2} \implies N \approx O(1/\epsilon^2) \end{cases}$$

For $p \neq 0$ or $p \neq 1$ the number of samples required i.e.e the number of circuit runs are $N \approx O(1/\epsilon^2)$. Thus, unless p is not close to 0 or p is not close to 1 we need to run the real Hadamard test $O(1/\epsilon^2)$ and run the imaginary hadamard test $O(1)$ i.e. a constant number of times to get the range of θ and thus to get the approximate value of θ to ϵ precision.

14.5 Kitaev's Method - for Quantum Phase Estimation

14.5.1 Idea of the Algorithm

The number of measurements needed to estimate θ to precision ϵ 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.,

$$\theta = (\theta_{d-1} \dots \theta_0)$$

Case: $d = 1$

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

$$p(1) = \frac{1}{2}(1 - \operatorname{Re}(e^{\pi i \theta_0})) = \frac{1}{2}(1 - \cos(\pi \theta_0))$$

Now,

$$\begin{cases} p(1) = 0 & \text{if } \theta_0 = 0 \\ p(1) = 1 & \text{if } \theta_0 = 1 \end{cases}$$

In, either case the result is deterministic, and one measurement is sufficient to determine the value of θ_0 .

Case: $d > 1$

Next, consider $\theta = 0.0\dots0\theta_0$. To determine the value of θ_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^{2^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 14.6 with $j = d - 1$ gives

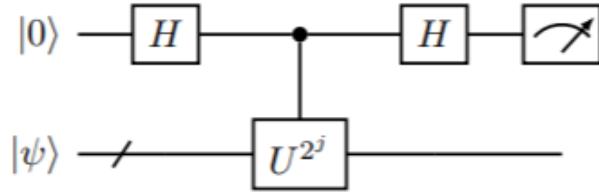


Figure 14.6: Kitaev's Method

$$\begin{aligned} p(1) &= \frac{1}{2}(1 - \text{Re}(\langle \psi | U^{2^{d-1}} | \psi \rangle)) \\ &= \frac{1}{2}(1 - \text{Re}(e^{2\pi i 2^{d-1}\theta})) \\ &= \frac{1}{2}(1 - \cos(2\pi 2^{d-1}\theta)) \end{aligned}$$

Now the value of $2^{d-1}0.0..0\theta_0 = \theta_0/2 = 0.\theta_0$ (in binary):

$$p(1) = \frac{1}{2}(1 - \cos(2\pi(0.\theta_0))) = \begin{cases} 0, & \theta_0 = 0 \\ 1, & \theta_0 = 1 \end{cases}$$

Thus using only one run we can determine $p(1)$ thus, we can find θ . If we get $|1\rangle$ upon measurement then $P(1) = 1$, thus $\theta_0 = 1 \implies \theta = 0.0\dots01$. If we get $|0\rangle$ upon measurement $p(1) = 0 \implies \theta_0 = 0 \implies \theta = 0.0\dots00$. Thus, the result is deterministic.

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 θ from a single number, we assume access to U^{2^j} , and estimate θ bit-by-bit.

$$2^j \theta = \theta_{d-1} \dots \theta_{d-j} \cdot \theta_{d-j-1} \dots \theta_0 = .\theta_{d-j-1} \dots \theta_0 \pmod{1}$$

for $j = 0, \dots, d-1$. Recall that $\cos(2\pi\theta_{d-1} \dots \theta_{d-j} \cdot \theta_{d-j-1} \dots \theta_0) = \cos(2\pi(\theta_{d-1} \dots \theta_{d-j} + 0 \cdot \theta_{d-j-1} \dots \theta_0)) = \cos(2\pi(0.\theta_{d-j-1} \dots \theta_0))$, since $\theta_{d-1} \dots \theta_{d-j}$ is an integer. Note that now upon running the circuit we would get $p(1)$ thus we can estimate $.\theta_{d-j-1} \dots \theta_0 \pmod{1}$. Thus, for $j = 0$ we need an accuracy of $\epsilon < 2^{-1}$ for θ to determine θ_{d-1} . Thus, using U , the number of samples/runs required are $N \approx O(\epsilon^2) \approx O(4)$.

For $j = 1$, we need an accuracy of $\epsilon < 1/2$ for θ to determine θ_{d-2} if we use U , but since we are going to use U^2 , the number of queries for u would be 2. Thus, in total we would require an accuracy of $\epsilon < 1/2$ and 2 queries to U each time giving total complexity as $O(2 \times 2^2)$.

For a general, j , we would need an accuracy of $\epsilon < 1/2$ and the number of queries to U would be 2^j . Thus, $O(2^j/\epsilon^2)$. Hence, then total complexity would be

$$\sum_{j=0}^{d-1} 4 \cdot 2^j = (4 \cdot 2^d) = O(2^d) = O(1/\epsilon)$$

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, as 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 θ and then the goal is to estimate its d bits. For simplicity of the analysis we assume that θ is exactly represented by d bits. We will extensively use the distance

$$|x|_1 = |x|_{mod1} = \min\{x \bmod 1, 1 - (x \bmod 1)\}$$

which is the distance on the unit circle.

14.5.2 Algorithm

- Now applying circuit in figure 14.6 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 α_j . This means that any perturbation must be due to the 5th digit in binary representation.

Example 14.5.1. If $2^j\theta = 0.11100 = 0.875 = 7/8$, 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 $0.0001 = 1/16$.

2. We then round $\alpha_j \pmod{1}$ by its closest 3-bit estimate denoted by β_j , i.e. β_j is taken from the set $\{0.000(=0), 0.001(=0.125=1/8), 0.010(=0.25), 0.011(=0.375=3/8), 0.100(=0.5), 0.101(=0.625), 0.110(=0.75), 0.111(=0.875)\}$.

Example 14.5.2. Consider the example of $2^j\theta = 0.11110(=15/16=0.9375)$, if $\alpha_j = 0.11101(=29/32=0.90625)$ then rounding $\alpha_j \pmod{1}$ to its closest 3-bit estimate is $\beta_j = 0.111(=7/8=0.875)$ from the set. But, if $\alpha_j = 0.11111(=31/32=0.96875)$, then $\beta_j = 0.000$ from the set, is not required to approximate.

Example 14.5.3. Another example is $2^j\theta = 0.11101$, if $\alpha_j = 0.11110(=15/16=0.9375)$ 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 α_j, β_j is not detrimental to the success of the algorithm.

3. Second, we perform post-processing. Start from $j = d - 3$, we can estimate $2^{d-3}\theta = 0.\theta_2\theta_1\theta_0 \pmod{1}$ to accuracy $1/16$, which recovers these three bits exactly. The values of these three bits will be taken from β_{d-3} directly. Then we proceed with the iteration for $j = d - 4, \dots, 0$, we assign

$$\theta_{d-j-1} = \begin{cases} 0, & |0\theta_{d-j-2}\theta_{d-j-3} - \beta_j| < 1/4 \\ 1, & |1\theta_{d-j-2}\theta_{d-j-3} - \beta_j| > 1/4 \end{cases}$$

There will be atmost 1 case satisfied always since $|x|_{mod1} < 1/2$ and they are separated by $1/2$. Thus, we recover $\theta = 0.\theta_{d-1} \dots \theta_0$ exactly. The total cost of Kitaev's method is measured in total number of queries to U is

$$\begin{aligned} & 1 + 2 + 2^2 + \dots + 2^{d-3} \\ &= \sum_{j=0}^{d-3} 2^j = 2^{d-2} - 1 = O(2^d) \\ &= O(1/\epsilon) \quad (\epsilon = 2^{-d}) \end{aligned}$$

Then total number of queries required is $\mathcal{O}(1/\epsilon)$. Note that because of the assumption that θ is exactly represented by d -bits. We have

$$|0.\theta_{d-1} \dots \theta_0|_{mod1} < 2^{-d} = \epsilon$$

j	$2^j\theta$	Possible β_j
0	0.11111	{0.111, 0.000}
1	0.1111	{0.111, 0.000}
2	0.111	{0.111}

Table 14.1: Results

14.5.3 Example

Consider $\theta = \theta_4\theta_3\theta_2\theta_1\theta_0 = 0.11111$. Here, $d = 5$. Running Kitaev's algorithm.

1. Run circuit for $j = 0, \dots, j = d - 3$ i.e. for $j = 0, 1, 2$. we can estimate $p(0)$ and thus get $2^j\theta$ upto an error of $1/16$. Consider the table 14.1 as shown given possible values of β_i after measuring α_j and doing a 3-bit estimate.
2. Start from $j = d - 3 = 2$. We have exact bits value $\alpha_j = 0.111$ and only one choice of $\beta_j = 0.111$.

Now for $j = 1$, using

$$\theta_{d-j-1} = \begin{cases} 0 & , |0.0\theta_{d-j-2}\theta_{d-j-3} - \beta_j|_{mod1} < 1/4 \\ 1 & , |0.1\theta_{d-j-2}\theta_{d-j-3} - \beta_j|_{mod1} < 1/4 \end{cases}$$

Therefore,

$$\theta_3 = \begin{cases} 0, & |0.0\theta_2\theta_1 - \beta_1|_{mod1} < 1/4 \\ 1, & |0.1\theta_2\theta_1 - \beta_1|_{mod1} < 1/4 \end{cases}$$

Now given possible choices of $\beta_j = \{0.111, 0.000\}$. Say we choose $\beta_j = 0.111$.

$$\theta_3 = \begin{cases} 0, & |0.011 - 0.111|_{mod1} = 7/8 - 3/8 = 1/2 > 1/4 \\ 1, & |0.111 - 0.111| = 0 < 1/4 \end{cases}$$

Thus, clearly we need to choose $\theta_3 = 1$ irrespective of value of β_1 we choose.

Similarly now for $j = 0$. Given that possible choice of $\beta_2\{0.111, 0.000\}$. Repeating the above steps we get, $\theta_4 = 1$. Thus, $\theta = 0.11111$ which recovers θ exactly.

14.5.4 Validity of Kitaev's Algorithm

Now we will inductively show why Kitaev's algorithm works. Again assume θ is exactly represented by d -bits. For $j = d - 3$, we know that $\theta_2\theta_1\theta_0$ can be recovered

exactly. Then assume $\theta_{d-j-2} \dots \theta_0$ have all been exactly computed, at step j we would like to determine the value of θ_{d-j-1} . From

$$|\alpha_j - 2^j \theta|_{mod1} < 1/16, \quad |\alpha_j - \beta_j|_{mod1} \leq 1/16$$

We know

$$|2^j \theta - \beta_j|_{mod1} < 1/8$$

Then

$$\begin{aligned} & |0.\theta_{d-j-1}\theta_{d-j-2}\theta_{d-j-3} - \beta_j|_{mod1} \\ & \leq |0.\theta_{d-j-1}\theta_{d-j-2}\theta_{d-j-3} - 2^j \theta|_{mod1} + |2^j \theta - \beta_j|_{mod1} \\ & \leq 1/16 + 1/8 < 1/4 \end{aligned}$$

The wrong choice of θ_{d-j-1} is denoted by $\tilde{\theta}_{d-j-1}$ then satisfies

$$\begin{aligned} & |0.\tilde{\theta}_{d-j-1}\theta_{d-j-2}\theta_{d-j-3} - \beta_j|_{mod1} \\ & \geq |0.\tilde{\theta}_{d-j-1}\theta_{d-j-2}\theta_{d-j-3} - 0.\theta_{d-j-1}\theta_{d-j-2}\theta_{d-j-3}|_{mod1} - |0.\theta_{d-j-1}\theta_{d-j-2}\theta_{d-j-3} - \beta_j|_{mod1} \\ & > 1/2 - 1/4 = 1/4 \end{aligned}$$

This proves the validity of the Kitaev's algorithm.

14.6 Quantum Phase Estimation using Quantum Fourier Transform

In Kitaev's method we use 1 ancilla qubit with d different circuits (of various circuit depths $U, U^2, \dots, U^{2^{d-1}}$, $j = 0$ to $j = d - 1$ i.e. d different circuits). Quantum Phase Estimation using Quantum Fourier transform uses 1 single Quantum circuit and d -ancilla qubits to store the phase information in the Quantum Computer.

To perform the estimation we assume that we have available black boxes (sometimes known as oracles) capable of preparing the state $|\psi\rangle$ and performing the controlled U^{2^j} operation, for suitable non-negative integers j . The use of black boxes indicate that the phase estimation procedure is not a complete quantum algorithm in its own right. Rather, you should think of phase estimation as a kind of ‘subroutine’ or ‘module’ that, when combined with other subroutines, can be used to perform interesting computation task. In specific applications of the phase estimation procedure, we will exactly do this, describing how these black box operations are to be performed, and combining them with the phase estimation procedure to do

genuinely useful tasks. For the moment though, we will continue to imagine them as black boxes.

The quantum phase estimation procedure uses two registers. The first register contains ($d = m$) bits initially in the state $|0\rangle$. How we choose m depends on two things: the number of digits of accuracy we wish to have in our estimation for θ , and with what probability we wish the phase estimation procedure to be successful. From now on, we will talk about the number of bits in the first register as m ($= d$) bits. The dependence of m on these quantities emerges naturally from the following analysis. The second register begins in the state $|\psi\rangle$, and contains as many qubits as is necessary to store $|\psi\rangle$.

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 dimensions $2^{m+n} \times 2^{m+n}$) 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 14.7:

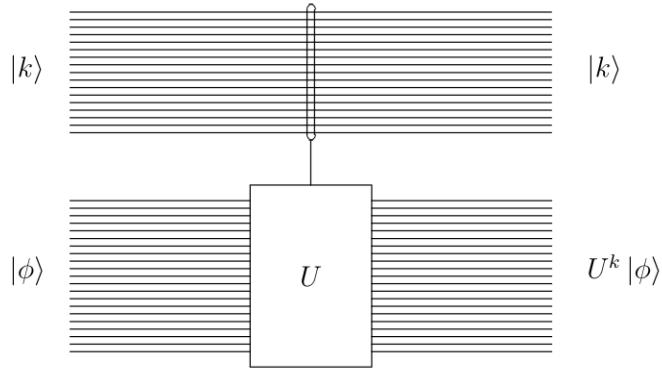


Figure 14.7: Controlled Unitary circuit

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 m , (which may be large or small).

14.6.1 Implementation of Controlled Unitary operation

The above circuit where $|k\rangle$ uses m bits to specify the number of times that U is to be applied on the remaining n qubits can be implemented as explained below. Let us now consider for a d -bit or m -bit ($m = d$) approximation of θ , we define a controlled U operation as

$$u = \sum_{j=0}^{2^m-1} |j\rangle\langle j| \otimes U^j$$

For example, when $m = 1$, $u = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U$. For a general m , it seems that we need to implement all 2^m different U^j . However, this is not necessary. Using the binary representation of integers $j = (j_{m-1} \dots j_0) = \sum_{i=0}^{m-1} j_i 2^i$ and hence $U^{\sum_{i=0}^{m-1} j_i 2^i} = \prod_{i=0}^{m-1} U^{j_i 2^i}$.

Also, note the property that

$$\begin{aligned} |j\rangle \langle j| &= |j_{m-1} \dots j_0\rangle \langle j_{m-1} \dots j_0| \\ &= (|j_{m-1}\rangle \otimes \dots \otimes |j_0\rangle)(\langle j_{m-1}| \otimes \dots \otimes \langle j_0|) \\ &= |j_{m-1}\rangle \langle j_{m-1}| \otimes \dots \otimes |j_0\rangle \langle j_0| \end{aligned}$$

Therefore similar to the operations in QFT,

$$\begin{aligned} u &= \sum_{j=0}^{2^m-1} |j\rangle \langle j| \otimes U^j \\ &= \sum_{j_{m-1}, \dots, j_0} (|j_{m-1}\rangle \langle j_{m-1}|) \otimes \dots \otimes (|j_0\rangle \langle j_0|) \otimes \left(U^{\sum_{i=0}^{m-1} j_i 2^i} \right) \\ &= \sum_{j_{m-1}, \dots, j_0} (|j_{m-1}\rangle \langle j_{m-1}|) \otimes \dots \otimes (|j_0\rangle \langle j_0|) \otimes \left(\prod_{i=0}^{m-1} U^{j_i 2^i} \right) \\ &= \tilde{\prod}_{i=0}^{m-1} \left(\sum_{j_i} |j_i\rangle \langle j_i| \otimes U^{j_i 2^i} \right) \\ &= \tilde{\prod}_{i=0}^{m-1} \left(|0\rangle \langle 0| \otimes I_n + |1\rangle \langle 1| \otimes U^{2^i} \right) \end{aligned}$$

$\tilde{\prod}$ means tensor product and \prod means normal matrix product (used in for the second register for showing matrix multiplication of $U^{j_i 2^i}$). This can be explained as follows.

Consider for $m = 2$

$$\begin{aligned}
u &= \sum_{j=0}^3 (|j\rangle\langle j| \otimes U^j) \\
&= |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U + |2\rangle\langle 2| \otimes U^2 + |3\rangle\langle 3| \otimes U^3 \\
&= |00\rangle\langle 00| \otimes U^0U^0 + |01\rangle\langle 01| \otimes UU^0 + |10\rangle\langle 10| \otimes U^0U^2 + |11\rangle\langle 11| \otimes UU^2 \\
&= (|0\rangle\langle 0| \otimes |0\rangle\langle 0|) \otimes I + (|0\rangle\langle 0| \otimes |1\rangle\langle 1|) \otimes U + \\
&\quad (|1\rangle\langle 1| \otimes |0\rangle\langle 0|) \otimes U^2 + (|1\rangle\langle 1| \otimes |1\rangle\langle 1|) \otimes U^3 \\
&= (|0\rangle\langle 0| \otimes I_n + |1\rangle\langle 1| \otimes U^2)(|0\rangle\langle 0| \otimes I_n + |1\rangle\langle 1| \otimes U) \\
&= (|0\rangle\langle 0| \otimes |0\rangle\langle 0| \otimes I_nI_n) + |0\rangle\langle 0| \otimes |1\rangle\langle 1| \otimes I_nU + (|1\rangle\langle 1| \otimes |0\rangle\langle 0|) \otimes U^2I_n + \\
&\quad (|1\rangle\langle 1| \otimes |1\rangle\langle 1| \otimes U^2U) \\
&= |0\rangle\langle 0| \otimes (|0\rangle\langle 0| \otimes I_n + |1\rangle\langle 1| \otimes U) + |1\rangle\langle 1| \otimes (|0\rangle\langle 0| \otimes U^2 + |1\rangle\langle 1| \otimes U^3) \\
&= (|0\rangle\langle 0| \otimes I_n) \otimes (|0\rangle\langle 0| \otimes I_n + |1\rangle\langle 1| \otimes U) + (|1\rangle\langle 1| \otimes U^2) \otimes (|0\rangle\langle 0| \otimes I_n + |1\rangle\langle 1| \otimes U) \\
&= (|0\rangle\langle 0| \otimes I_n + |1\rangle\langle 1| \otimes U^2) \otimes (|0\rangle\langle 0| \otimes I_n + |1\rangle\langle 1| \otimes U) \\
&= \prod_{i=0}^3 (|0\rangle\langle 0| \otimes I_n + |1\rangle\langle 1| \otimes U^{2^i}) \\
&= \prod_{i=0}^{m-1} (|0\rangle\langle 0| \otimes I_n + |1\rangle\langle 1| \otimes U^{2^i})
\end{aligned}$$

Thus,

$$u = \prod_{i=0}^{m-1} (|0\rangle\langle 0| \otimes I_n + |1\rangle\langle 1| \otimes U^{2^i})$$

is the implementation of controlled U operation. Now consider the circuit shown in figure 14.8. **Analysis:**

Consider $u = \sum_{j=0}^{2^m-1} |j\rangle\langle j| \otimes U^j$. Now, as input j will be in one of the basis states say $|p\rangle$, $p \in \{0, \dots, 2^m - 1\}$. Then,

$$\begin{aligned}
u|p\rangle &= \left(\sum_{j=0}^{2^m-1} |j\rangle\langle j| \otimes U^j \right) (|p\rangle \otimes |\psi\rangle) \\
&= \sum_{j=0}^{2^m-1} |j\rangle\langle j| |p\rangle \otimes U^j |\psi\rangle \\
u|p\rangle &= |p\rangle \otimes U^p |\psi\rangle
\end{aligned}$$

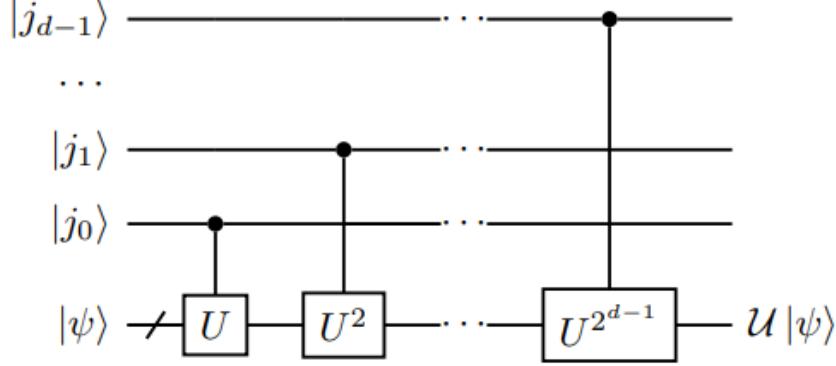


Figure 14.8: Implementation of Controlled Unitary operation

Thus, u takes input a basis state $|p\rangle$ and applies Unitary matrix U^p to $|\psi\rangle$.

Thus, we need to apply U^p corresponding to basis state $|p\rangle$. Now from figure, we get that corresponding to $|j_{m-1}\dots j_0\rangle = |j\rangle$, we apply a controlled U operations as U^{2^j} . From figure, thus corresponding to j_k we apply U^{2^k} , $k \in \{0, \dots, m-1\}$.

This is advantageous when U^j can be implemented at a cost-independent of j . Example if $U = R_2(\theta)$ is a single qubit rotation. This is exponentially better than the direct implementation of u .

Example 14.6.1. Additional insight into the circuit may be obtained by showing that the effect of the sequence of controlled U operations like that is to take the state $|j\rangle|u\rangle$ to $|j\rangle U^j|u\rangle$. (Note that this does not depend on $|u\rangle$ being an eigenstate of U).

From the figure 14.8, we can see that

$$\begin{aligned} |j\rangle|u\rangle &\rightarrow |j\rangle(U^{2^0})^{j_0}(U^{2^1})^{j_1}\dots(U^{2^{n-1}})^{j_{n-1}}|u\rangle \\ &\rightarrow |j\rangle U^{j_02^0+j_12^1+\dots+j_{n-1}2^{n-1}}|u\rangle \\ &\rightarrow |j\rangle U^j|u\rangle \end{aligned}$$

14.6.2 Implementation of QPE Circuit

Consider the following circuit diagram 14.9:

Note that we have used $d = m$ interchangeably. 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 (which is actually the controlled

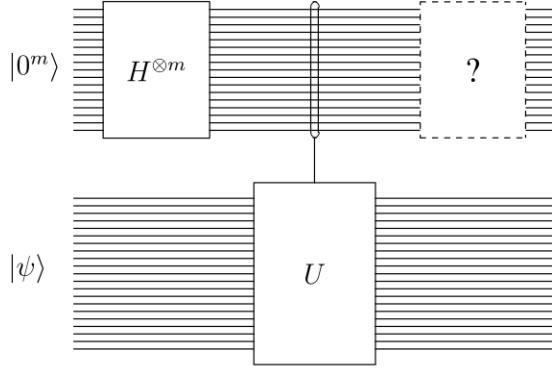


Figure 14.9: circuit

U operation as explained before) 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. (Note that here it is not a necessary need $|\psi\rangle$ to be an eigenvector as for any vector $|\phi\rangle = \sum_i \alpha_i |\psi_i\rangle$, where $|\psi_i\rangle$ are eigenvectors of $|U\rangle$ using the spectral decomposition of U .) 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:

$$\left(\frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i k \theta} |k\rangle \right) \otimes |\psi\rangle$$

Another way to think about this is that by applying a Hadamard transform to the first register followed by an application of controlled-U operations to the second register, with U raised to successive powers of two. The final state of the register is easily seen to be:

$$\frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i \theta k} |k\rangle |\psi\rangle = \left(\frac{|0\rangle + e^{2\pi i 2^{d-1} \theta} |1\rangle}{\sqrt{2}} \right) \left(\frac{|0\rangle + e^{2\pi i 2^{d-2} \theta} |1\rangle}{\sqrt{2}} \right) \dots \left(\frac{|0\rangle + e^{2\pi i 2^0 \theta} |1\rangle}{\sqrt{2}} \right) \otimes |\psi\rangle$$

So, if we discard the last n qubits we are left with the state:

$$\frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i k \theta} |k\rangle$$

The second stage of phase estimation is to apply the inverse quantum Fourier transform on the first register. This is obtained by reversing the circuit for the quantum Fourier transform as shown in the previous chapter and can be done in $\theta(m^2)$ steps. The third and final stage of phase estimation is to read out the state of the first register by doing a measurement in the computational basis. This provides a pretty good estimate of θ . An overall schematic of the algorithm is shown in the figure 14.10.

A simple case: Exact θ

Recall that our goal is to approximate θ . Suppose that θ happens to have a special form:

$$\theta = \frac{j}{2^m}$$

for some integer $j \in \{0, \dots, 2^m - 1\}$. (This makes θ take only certain distinct values thus, in general, it is not correct to assume that θ 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 $\omega = e^{2\pi i/2^m}$, which are the roots $N = 2^m$ roots of unity, 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 θ 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 ω 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 vectors $|\phi_j\rangle$ to determine the columns of D :

$$QFT_{2^m} = (|\phi_0\rangle \quad |\phi_1\rangle \quad \dots \quad |\phi_{2^m-1}\rangle)$$

$$QFT_{2^m} = \frac{1}{\sqrt{2^m}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{2^m-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(2^m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{2^m-1} & \omega^{2(2^m-1)} & \dots & \omega^{(2^m-1)(2^m-1)} \end{pmatrix}$$

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, $QFTQFT^\dagger = QFT^\dagger QFT = I \implies QFT^\dagger = QFT^{-1}$). $QFT^\dagger = QFT^{-1}$ since the matrix is also Unitary). This is the *Discrete Fourier Transform* matrix. 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 14.10. Thus, measuring the first m qubits and dividing by 2^m tell us precisely the value of θ . **But this assumes that:**

$$U|\psi\rangle = e^{2\pi i j / 2^m} |\psi\rangle$$

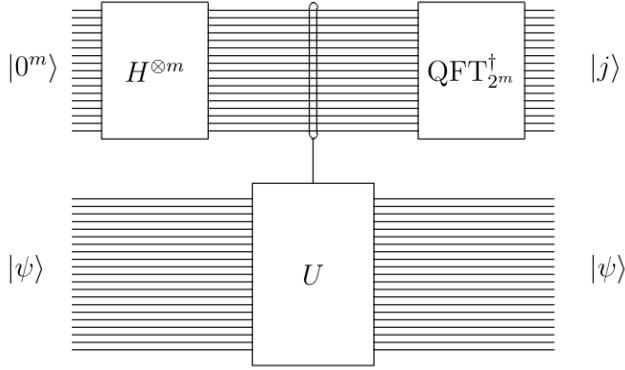


Figure 14.10: QPE circuit

We still need to prove this for the more general case that θ 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 θ**). The two major issues to deal with are:

1. What can be said about the measurement in the case that θ 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 θ

From the above analysis we know that for an arbitrary value of θ , the state of the above circuit immediately before the inverse of the QFT is applied is

$$\frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi i k \theta} |k\rangle |\psi\rangle$$

Recall that,

$$QFT_{2^m}^\dagger |k\rangle = \frac{1}{\sqrt{2^m}} \sum_{j=0}^{2^m-1} e^{-2\pi i j k / 2^m} |j\rangle$$

Now, 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 showed 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 θ 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 ϵ 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 lower bound on p_j as follows: Let

$$a = |e^{2\pi i(2^m\theta - j)} - 1| = |e^{2\pi i\epsilon 2^m} - 1| = |\cos(2\pi\epsilon 2^m) + i \sin(2\pi\epsilon 2^m) - 1|$$

$$b = |e^{2\pi i(\theta - j/2^m)} - 1| = |e^{2\pi i\epsilon} - 1| = |\cos(2\pi\epsilon) + i \sin(2\pi\epsilon) - 1|$$

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 of the circle at origin in the complex plane as shown in figure 14.11: Now from the figure recall that the ratio of the minor arc length to the chord, for $\theta \in [-\pi, \pi]$ 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, using complex place geometry, from the figure 14.11 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 the ratio is always greater than 1. Thus,

$$\frac{2\pi|\epsilon|}{b} \geq 1 \implies b \leq 2\pi|\epsilon|$$

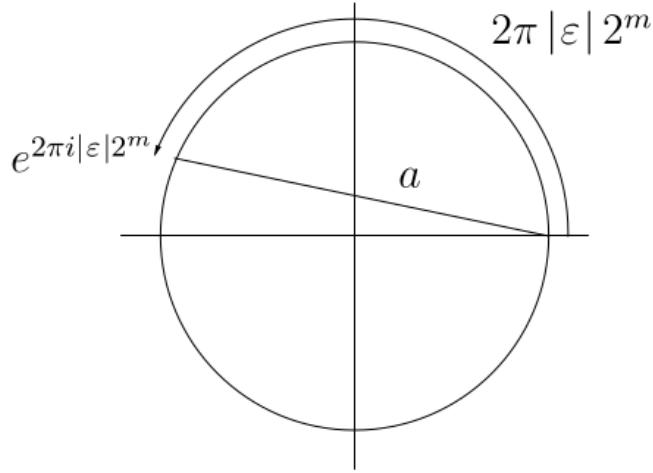


Figure 14.11: Lower bound of a

Thus, we can say that the probability p_j is lower bounded by:

$$\begin{aligned} p_j &= \frac{1}{2^{2m}} \frac{a^2}{b^2} \geq \frac{1}{2^{2m}} \frac{(4|\epsilon|2^m)^2}{(2\pi|\epsilon|)^2} = \frac{1}{2^{2m}} \frac{16|\epsilon|^2 2^{2m}}{4\pi^2|\epsilon|^2} = \frac{4}{\pi^2} = 0.405284 \\ \implies p_j &\geq 0.405284 \end{aligned}$$

Thus, the probability is atleast 40.5% of measuring the value of j to which θ is close to that $j/2^m$. This is the probability that every single one of the bits you measure is correct, so that your approximation to θ 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 ϵ with $\frac{\alpha}{2^m} \leq |\epsilon| < 1/2$. Here α 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 14.11, we can say that $a \leq 2$ (because the maximum possible $|a| = 2$ when it is the diameter of the circle in the complex plane) and $b \geq 4|\epsilon|$. Now we have,

$$p_j \leq \frac{4}{2^{2m}(4|\epsilon|)^2} = \frac{1}{4(2^m|\epsilon|)^2} = \frac{1}{4\alpha^2}$$

This implies that highly inaccurate results are very unlikely. For example, even 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 $|\epsilon|$ (for higher values of α), the probability of obtaining the corresponding value of j quickly becomes very small.

Let b be the integer in the range 0 to $2^m - 1$ such that $b/2^t = 0.b_{t-1}\dots b_1$ is the best m -bit approximation to θ which is less than θ . That is, the difference $\delta = \theta - b/2^t$ between θ and $b/2^t$ satisfies $0 \leq \delta \leq 2^{-t}$. We aim to show that the observation at the end of phase estimation procedure produces a result which is close to b , and thus enables us to estimate θ accurately, with high probability. Apply the inverse quantum Fourier transform produces the state

$$\frac{1}{2^t} \sum_{k,l=0}^{2^t-1} e^{-2\pi\imath kl} 2^t e^{2\pi\imath \theta k} |l\rangle$$

Let α_l be the amplitude of $|(b+l)(\text{mod } 2^t)\rangle$,

$$\alpha_l = \frac{1}{2^t} \sum_{k=0}^{2^t-1} \left(e^{2\pi\imath(\theta-(b+1)/2^t)} \right)^k$$

This is the sum of a geometric series, so

$$\begin{aligned} \alpha_l &= \frac{1}{2^t} \left(\frac{1 - e^{2\pi\imath(2^t\theta-(b+l))}}{1 - e^{2\pi\imath(\theta-(b+l)/2^t)}} \right) \\ &= \frac{1}{2^t} \left(\frac{1 - e^{2\pi\imath(2^t\delta-l)}}{1 - e^{2\pi\imath(\delta-l/2^t)}} \right) \end{aligned}$$

Suppose the outcome of the final measurement is m . We aim to bound the probability of obtaining a values of m such that $|m - b| > e$, where e is a positive integer characterizing our desired tolerance to error. The probability of observing such an m is given by

$$p(|m - b| > e) = \sum_{-2^{t-1} < l \leq -(e+1)} |\alpha_l|^2 + \sum_{e+1 \leq l \leq 2^{t-1}} |\alpha_l|^2$$

But for any real θ , $|1 - \exp(i\theta)| \leq 2$, so

$$|\alpha_l| \leq \frac{1}{2^{t+1}(\delta - l/2^t)}$$

Combining the two equation gives

$$p(|m - b| > e) \leq \frac{1}{4} \left(\sum_{l=-2^{t-1}+1}^{-e+1} \frac{1}{l^2} + \sum_{l=e+1}^{2^t-1} \frac{1}{(l - 2^t\delta)^2} \right)$$

Recalling that $0 \leq 2^t\delta \leq 1$, we obtain

$$\begin{aligned} p(|m - b| > e) &\leq \frac{1}{4} \left(\sum_{l=-2^{t-1}+1}^{-e+1} \frac{1}{l^2} + \sum_{l=e+1}^{2^t-1} \frac{1}{(l - 2^t\delta)^2} \right) \\ &\leq \frac{1}{2} \sum_{l=e}^{2^t-1-1} \frac{1}{l^2} \\ &\leq \frac{1}{2} \int_{e-1}^{2^t-1-1} \frac{1}{l^2} dl \\ &= \frac{1}{2(e-1)} \end{aligned}$$

Suppose we wish to approximate θ to an accuracy 2^{-n} , that is, we choose $e = 2^{t-n} - 1$. by making use of $t = n + p$ qubits in the phase estimation algorithm we see from the results that the probability of obtaining an approximation correct to this accuracy is at least $1 - 1/2(2^p - 2)$. Thus to successfully obtain θ accurate to n bits with probability of success at least $1 - \epsilon$ we choose

$$t = n + \lceil \log \left(2 + \frac{1}{2\epsilon} \right) \rceil$$

In order to make use of the phase estimation algorithm, we need to be able to prepare an eigenstate of $|\psi\rangle$ of U . What if we do not known how to prepare such an eigenstate? Suppose that we prepare some other state $|\varphi\rangle$ in place of $|\psi\rangle$. Expanding the state in terms of the eigenstates $|\psi_i\rangle$ of U gives $|\varphi\rangle = \sum_i c_i |\psi_i\rangle$. Intuitively, the result of running the phase estimation algorithm will be to give as output a state close to $\sum_i c_i |\tilde{\theta}_i\rangle |\psi_i\rangle$, where $\tilde{\theta}_i$ is a pretty good approximation to the phase θ_i . Therefore, we expect that reading out the first register will give us a good

approximation to θ_i , where i is chosen at random with probability $|c_i|^2$. Making this argument rigorous is as shown. This procedure allow us to avoid preparing a (possibly unknown) eigenstate, at the cost of introducing some additional randomness into the algorithm.

Thus, to have a better result than a $4/\pi^2$ probability of obtaining an approximation of θ upto k bits of precision. Set $m = k + 2$ 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.

Another way of intuition as to why phase estimation works, suppose θ may be expressed exactly in d bits, as $\theta = 0.\theta_{d-1} \dots \theta_0$. Then the state resulting from the first stage of phase estimation may be rewritten

$$\frac{1}{\sqrt{2^d}}(|0\rangle + e^{2\pi i 0.\theta_0} |1\rangle)(|0\rangle + e^{2\pi i 0.\theta_1\theta_0} |1\rangle) \dots (|0\rangle + e^{2\pi i 0.\theta_{d-1}\theta_{d-2} \dots \theta_1\theta_0} |1\rangle)$$

The second stage of phase estimation is to apply the inverse quantum Fourier transform. But comparing the previous equation with the product form for the Fourier transform, we see that the output state from the second stage is the product state $|\theta_{d-1} \dots \theta_0\rangle$. A measurement in the computational basis therefore gives θ exactly.

Summarizing, the phase estimation algorithm allows one to estimate the phase θ of an eigenvalue of a unitary operator U , given the corresponding eigenvector $|\psi\rangle$. An essential feature at the heart of this problem is the ability of the inverse Fourier transform to perform the transformation

$$\frac{1}{\sqrt{2^m}} \sum_{j=0}^{2^m-1} e^{2\pi i \theta j} |j\rangle |\psi\rangle \rightarrow |\tilde{\theta}\rangle |\psi\rangle$$

where $|\tilde{\theta}\rangle$ denotes a state which is a good estimator for θ when measured.

14.7 Special Case: Superposition of Eigenvectors

Suppose that, instead of being given an eigenvector of U , we are given a superposition of eigenvectors. Recall that since unitary matrix is a normal matrix it's eigen vectors

Algorithm 2 Quantum Phase Estimation

- 1: **Input:** (1) A black box which performs a controlled- U^j operation, for integer j , (2) an estimate $|\psi\rangle$ of U with eigenvalue $e^{2\pi\iota\theta}$, and (3) $t = n + \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ qubits initialized to $|0\rangle$.
 - 2: **Output:** An n -bit approximation $\tilde{\theta}$ to θ .
 - 3: **Runtime:** $O(t^2)$ operations and one call to controlled- U^j black box. Succeeds with probability at least $1 - \epsilon$.
 - 4: **Procedure:**
 - 5: **Initial State:** $|0\rangle |\psi\rangle$
 - 6: **Create Superposition** $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle |\psi\rangle$
 - 7: **Apply Black box:** $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle U^j |\psi\rangle$
 - 8: **Result of black box:** $= \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} e^{2\pi\iota j\theta} |j\rangle |\psi\rangle$
 - 9: **Apply Inverse Fourier Transform** $\rightarrow |\tilde{\theta}\rangle |\psi\rangle$
 - 10: **measure first register** $\rightarrow \tilde{\theta}$
-

will span the entire \mathbb{C}^N . Consider that we are given some random state $|\psi\rangle$ then, we can write this as

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |\lambda_i\rangle$$

where $\alpha_i \in \mathbb{C}$ and $\{|\lambda_i\rangle\}_{i=0}^{2^n-1}$ are the eigenvectors and λ_i are the corresponding eigen values of U . Then, upon running the phase estimation algorithm for this state, treating the entire circuit as a unitary matrix at the end we would reach the state

$$\sum_{j=0}^{2^m-1} \left(\sum_{i=0}^{2^n-1} \sum_{k=0}^{2^m-1} \frac{1}{2^m} \alpha_i e^{2\pi\iota(k\lambda_i - jk/2^m)} \right) |j\rangle |\lambda_i\rangle$$

Thus, the probability of achieving a state $|j\rangle$ depends on i as well,

$$\begin{aligned} p_j &= \left| \sum_{i=0}^{2^n-1} \frac{1}{2^m} \sum_{k=0}^{2^m-1} \alpha_i e^{2\pi\iota k(\lambda_i - j/2^m)} \right|^2 \\ &= \frac{1}{2^{2m}} \left| \sum_{i=0}^{2^n-1} \sum_{k=0}^{2^m-1} \alpha_i e^{2\pi\iota k(\lambda_i - j/2^m)} \right|^2 \\ &= \frac{1}{2^{2m}} \left| \sum_{i=0}^{2^n-1} \alpha_i \sum_{k=0}^{2^m-1} e^{2\pi\iota k(\lambda_i - j/2^m)} \right|^2 \end{aligned}$$

Thus, the probability of getting a state $|j\rangle$ depends on the distribution of eigenvalues over the unit circle, our accuracy i.e. approximation of m bits chosen as well as the initial superposition $|\psi\rangle$ overlap with one of the eigen vectors $|\lambda_i\rangle$ i.e. the coefficient α_i .

14.8 Complexity Analysis

In order to apply QPE we have assumed that

1. $|\psi_0\rangle$ is an eigenstate.
2. θ_0 has a d-bit binary representation

In general practical calculations, neither conditions can be exactly satisfied, and we need to analyze the effect on the error of the QPE. Recall the discussion, we assume that the only sources of errors are at the mathematical level (instead of noisy implementation of quantum devices). In this context, the error can be due to an inexact eigenstate $|\psi\rangle$ or Monte Carlo errors in the readout process due to the probabilistic nature of the measurement process. In this section, we relax these constraints and study what happens when the conditions are not exactly met. We assume U has the eigen decomposition

$$U |\psi_j\rangle = e^{i2\pi\theta_j} |\psi_j\rangle$$

Without loss of generality we assume $0 \leq \theta_0 \leq \theta_1 \dots \leq \theta_{N-1} < 1$. We are interested in using QPE to find the value of θ_0 .

We first relax the condition (1)i.e., assume all θ'_i s have an exact d-bit binary representation, but the quantum state is given by a linear combination

$$|\phi\rangle = \sum_{k=0}^{N-1} c_k |\psi_k\rangle$$

Here, the overlap $p_0 = |\langle \phi | \psi_0 \rangle|^2 = |c_0|^2 < 1$.

Applying the QPE circuit in figure 14.10 to $|0^t\rangle |\phi\rangle$ with $t = d$, and measure the ancilla qubits, we obtain the binary representation of θ_0 with probability p_0 . Furthermore, the system register returns the eigenstate $|\psi_0\rangle$ with probability p_0 . Of course, in order to recognize that θ_0 is the desired phase, we need some a priori knowledge of the location of θ_0 . e.g. $\theta_0 \in (a, b)$ and $\theta_i > b$ for all $i \neq 0$. It would be desirable to relax both conditions (1) and (2). However, the analysis can be rather involved and additional assumptions are needed.

The above analysis applies to the ideal case, where θ can be written exactly with a d bit binary expansion. What happens when this is not the case? It turns out that the procedure we have described will produce a pretty good approximation to θ with high probability, as shown earlier.

For now, to simplify the analysis, we focus on the case that the only condition (2) is violated, i.e. θ_0 cannot be exactly represented by a d -bit number, and we need to apply the QPE circuit to an initial state $|0^t\rangle|\phi\rangle$ with $t > d$. The exact relation between t and the desired accuracy d will be determined later. Similar to the equation of QPE before, we obtain the state

$$\begin{aligned} |0^t\rangle|\psi_0\rangle &\rightarrow \sum_{k'=0}^{2^m-1} \left(\frac{1}{T} \sum_{j=0}^{T-1} e^{\iota 2\pi j(\theta_0 - \frac{k'}{T})} \right) |k'\rangle|\psi_0\rangle \\ &= \sum_{k'} \gamma_{0,k'} |k'\rangle|\psi_0\rangle \end{aligned}$$

Here

$$\gamma_{0,k'} = \frac{1}{T} \sum_{j=0}^{T-1} e^{\iota 2\pi j(\theta_0 - \frac{k'}{T})} = \frac{1}{T} \frac{1 - e^{\iota 2\pi T(\theta_0 - \tilde{\theta}_{k'})}}{1 - e^{\iota 2\pi(\theta_0 - \tilde{\theta}_{k'})}}, \quad \tilde{\theta}_{k'} = \frac{k'}{T}$$

Therefore if θ_0 has an exact d -bit representation, i.e. $\theta_0 = \tilde{\theta}_{k'_0}$ for any k' , note that $e^{\iota 2\pi x}$ is a periodic function with period 1, we can only determine the value of $x \bmod 1$. Therefore we use the periodic distance. In terms of the phase, we would like to find k'_0 such that

$$|\theta_0 - \tilde{\theta}_{k'_0}| < \epsilon$$

Here $\epsilon = 2^{-d} = 2^{t-d}/T$ is the precision parameter. In particular, for any k' we have

$$|\theta_0 - \tilde{\theta}_{k'}|_1 \leq \frac{1}{2}$$

Using the relation that for any $\theta \in [-\pi, \pi]$,

$$|1 - e^{\iota\theta}| = \sqrt{2(1 - \cos\theta)} = 2|\sin(\theta/2)| \geq \frac{2}{\pi}|\theta|,$$

we obtain

$$|\gamma_{0,k'}| \leq \frac{2}{T \frac{2}{\pi} 2\pi |\theta_0 - \tilde{\theta}_{k'}|_1} = \frac{1}{2T |\theta_0 - \tilde{\theta}_{k'}|_1}$$

Let k'_0 be the measurement outcome, which can be viewed as a random variable. The probability of obtaining some $\tilde{\theta}_{k'}$ that is at least ϵ distance away from θ_0 is

$$\begin{aligned} P(|\theta_0 - \tilde{\theta}_{k'_0}|_1 \geq \epsilon) &= \sum_{|\theta_0 - \tilde{\theta}_{k'}|_1 \geq \epsilon} |\gamma_{0,k'}|^2 \\ &\leq \sum_{|\theta_0 - \tilde{\theta}_{k'}|_1 \geq \epsilon} \frac{1}{4T^2 |\theta_0 - \tilde{\theta}_{k'}|^2} \\ &\leq \frac{2}{4T} \int_{\epsilon}^{\infty} \frac{1}{x^2} dx + \frac{2}{4T^2 \epsilon^2} = \frac{1}{2T\epsilon} + \frac{1}{2(T\epsilon)^2} \end{aligned}$$

set $t = d = \lceil \log_2 \delta^{-1} \rceil$, then $T\epsilon = 2^{t-d} \geq \delta^{-1}$. Hence for any $0 < \delta < 1$, the failure probability

$$P(|\theta_0 - \tilde{\theta}_{k'_0}|_1 \geq \epsilon) \leq \frac{\delta + \delta^2}{2} \leq \delta$$

In other words, in order to obtain the phase θ_0 to accuracy $\epsilon = 2^{-d}$ worth a success probability at least $1 - \delta$, we need $d + \lceil \log_2 \delta^{-1} \rceil$ ancilla qubits to store the value of the phase. On top of that, the simulation time needs to be $T = (\epsilon\delta)^{-1}$.

Remark. (Quantum median method). One problem with QPE is that in order to obtain a success probability $1 - \delta$, we must use $\log_2 \delta^{-1}$ ancilla qubits, and the maximal simulation time also needs to be increase by a factor of δ^{-1} . The increase of the maximal simulation time is particularly undesirable since it increases the circuit depth and hence the required coherence time of the quantum device. When $|\psi\rangle$ is an exact eigenstate, this can be improved by the median method, which uses $\log \delta^{-1}$ copies of the result from QPE without using ancilla qubits or increasing the circuit depth. When $|\psi\rangle$ is a linear combination of eigenstate, the problem of the aliasing effect becomes more difficult to handle. one possibility is to generalize the median method into the quantum median method, which uses classical arithmetic to evaluate the median using a quantum circuit. To reach success probability $1 - \delta$, we will need $\log_2 \delta^{-1}$ ancilla qubits, but the maximal simulation time does not need to be increased.

Why is phase estimation interesting? For its own sake, phase estimation solves a problem which is both non-trivial and interesting from a physical point of view: how to estimate the eigenvalue associated with a given eigenvector for a unitary operator. Its real use, though, comes from the observation that other interesting problems can be reduced to phase estimation, as will be shown in subsequent chapters.

Example 14.8.1. Suppose the phase estimation algorithm takes the state $|0\rangle |u\rangle$ to the state $|\tilde{\phi}_u\rangle |u\rangle$, so that given the input $|0\rangle (\sum_u c_u |u\rangle)$, the algorithm outputs

$\sum_u c_u |\tilde{\phi}_u\rangle |u\rangle$. Show that if t is chosen according to the equation, then the probability for measuring ϕ_u accuracy to n bits at the conclusion of the phase estimation algorithm is at least $|c_u|^2(1 - \epsilon)$.

With probability $|c_u|^2$ we will be measuring ϕ_u for the state $|u\rangle$. If t is of the form of $t = n + \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ each $\tilde{\phi}_u$ is accurate to n bits of ϕ_u with probability $1 - \epsilon$. Hence, the probability of measuring ϕ_u accurate to n bits is $|c_u|^2(1 - \epsilon)$.

Example 14.8.2. Let U be a unitary transform with eigenvalue ± 1 , which acts on a state $|\psi\rangle$. Using the phase estimation procedure, construct a quantum circuit to collapse $|\psi\rangle$ into one or the other of the two eigenspace of U , giving also a classical indicator as to which space the final state is in.

Consider the circuit shown in the figure 14.1. 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:

$$\begin{aligned} (H \otimes I)\frac{1}{\sqrt{2}}(|0\rangle |\psi\rangle + |1\rangle U |\psi\rangle) &\rightarrow \frac{1}{\sqrt{2}}(H|0\rangle |\psi\rangle + H|1\rangle U |\psi\rangle) \\ \frac{1}{\sqrt{2}}(H|0\rangle |\psi\rangle + H|1\rangle U |\psi\rangle) &= \frac{1}{2}(|0\rangle (|\psi\rangle + U|\psi\rangle) + |1\rangle (|\psi\rangle - U|\psi\rangle)) \end{aligned}$$

Now, using the fact that $|\psi\rangle = \alpha_+ |u_+\rangle + \alpha_- |u_-\rangle$, and substituting, the above simplifies to,

$$|0\rangle \alpha_+ |u_+\rangle + |1\rangle \alpha_- |u_-\rangle$$

Thus, the quantum circuit shows that it collapses to $|u_+\rangle$ if the measurement of the first qubit is $|0\rangle$ and to the second qubit if the measurement of the first qubit is $|1\rangle$. Here, we have expanded $|\psi\rangle$ into the eigenbasis of U with eigenvectors $|u_+\rangle$ corresponding to eigenvalue $+1$ and $|u_-\rangle$ with the eigenvalue -1 , the α_+ and α_- are the coefficients of the vectors. The probability of measuring the first qubit in state $|0\rangle$ is $|\alpha_+|^2$ and the probability of measuring the second qubit in the state $|1\rangle$ is $|\alpha_-|^2$. After measurement the state collapses to either $|0\rangle |u_+\rangle$ or $|1\rangle |u_-\rangle$ depending on whether $|0\rangle$ or $|1\rangle$ is measured in the first qubit and thus after measurement the state

of the second qubit is either $|u_+\rangle$ or $|u_-\rangle$ respectively. Hence, we have constructed a quantum circuit which collapses $|\psi\rangle$ to one or the other two eigenspaces with the classical indicator as the first qubit collapsing to either $|0\rangle$ or $|1\rangle$ corresponding to $|u_+\rangle$ or $|u_-\rangle$ state in the final state.

14.9 Implementation using Qiskit

We will now implement the Quantum Phase Estimation using Quantum Fourier Transform. Consider the following Unitary we wish to implement.

$$R|1\rangle = \begin{pmatrix} 1 & 0 \\ 0 & e^{i2\pi\theta} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = e^{i2\pi\theta}|1\rangle$$

with $\theta = 0.5 + \frac{1}{2^d} = 0.100\dots01$ (d bits in total). Say for example we take $d = 4$ and the exact value is $\theta = 0.5625$. Now we perform the Hadamard Test as shown using the following code.

Chapter 15

Order Finding

15.1 Introduction

Now that we have discussed the phase estimation technique, it is time to apply it to an interesting computational problem. The problem is called Order Finding, which is one step from the factoring problem. This will be the quantum part of the Shor's algorithm - the rest is completely classical after this. In the next chapter we will move on to Shor's factoring algorithm. We will now review some basics about divisors, common divisors, and the greatest common divisor of two integers.

The fast quantum algorithms for factoring and order-finding requires are interesting for at least three reasons. first, and most important, they provide evidence for the idea that quantum computers may be inherently more powerful than classical computers, and provide a credible challenge to the strong Church-Turing Thesis. Second, both problems are of sufficient intrinsic worth to justify interest in any novel algorithm, be it classical or quantum. Third, and most important from a practical standpoint, efficient algorithms for order finding and factoring can be used to break the RSA public-key cryptosystem.

Of course, we know that factoring a number x (a positive integer) is about finding its divisors (where the divisors are the numbers that divide x). If we have two numbers, x and y , then the common divisors of x and y are the numbers that divide both of them. For example, for the numbers 28 and 42 the common divisors are : 1, 2, 7, and 14 (4 is not a common divisor because, although 4 divides 28, it does not divide 42).

Definition 15.1.1. (greatest common divisor (GCD)). For two numbers x and y , their greatest common divisor is the largest number that divides both x and y . This is commonly denoted as $gcd(x, y)$.

The GCD of 28 and 42 is 14. The GCD of 16 and 21 is 1, since there is no larger integer that divides both of them.

Definition 15.1.2. (relatively prime) We say that the numbers x and y are relatively prime if $\gcd(x, y) = 1$. That is, they have no common divisor, except trivial divisor 1.

Note that 16 and 21 are not prime but they are relatively prime.

Superficially, the problem of finding common divisors resembles the problem of finding divisors (which is the factoring problem). If x and y are n -bit numbers then a brute-force search would have to check among exponentially many possibilities.

In fact, the problem of finding greatest common divisors is considerably easier than factoring. It has been known for a long time that there is an efficient algorithm for computing GCDs. By long time, I mean approximately 2300 years! That's how long ago Euclid published his algorithm (now known as the Euclidean algorithm). Of course, there were no computers back then, but Euclid's algorithm could be carried out by a hand calculation and it requires $\mathcal{O}(n)$ arithmetic operations for n -digit numbers. This translates into a gate cost of $\mathcal{O}(n^2 \log n)$. Let's remember the Euclid's GCD algorithm, because we're going to make use of it.

Now, I'd like to briefly review a few more properties of numbers. Let $N \geq 2$ be an integer that's not necessarily prime. Recall that $\mathbb{Z}_m = \{0, 1, 2, \dots, N - 1\}$ and

$$\mathbb{Z}_N^* = \{x \in \mathbb{Z}_N : x \text{ has a multiplicative inverse mod } N\}$$

where the latter is a group, with respect to multiplication mod N . For example,

$$\mathbb{Z}_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$$

It turns out that x has a multiplicative inverse mod N if and only if $\gcd(x, N) = 1$. For example, you can see that 3, 6, and 7 (and several other number in \mathbb{Z}_{21}) are excluded from \mathbb{Z}_{21}^* because they have common factors with 21.

Example 15.1.1. Show that the order of $x = 5$ modulo $N = 21$ is 6.

Now, let's take an element of \mathbb{Z}_{21}^* and list all of its powers. Suppose we pick 5. The powers of 5 ($5^0, 5^1, 5^2, \dots$) are

$$1, 5, 4, 20, 16, 17, 1, 5, 4, 20, 16, 17, 1, 5, 4, 20, 16, \dots$$

Notice that it's periodic ($5^6 = 1$, and then subsequent powers repeat the same sequence). The period of the sequence is 6.

If we pick 4 instead of 5 then the period of the sequence is different. The powers of 4 are: 1, 4, 16, 1, 4, 16, ... so the period is 3.

For positive integers x and N , $x < N$, with no common factors, the order of x modulo N is defined to be the least positive integer r , such that $x^r \equiv 1 \pmod{N}$.

Definition 15.1.3. (**order of $a \in \mathbb{Z}_N^*$**) For any $a \in \mathbb{Z}_N^*$ define the order of a modulo N (denoted as $\text{ord}_N(a)$) as the minimum positive r such that $a^r \equiv 1 \pmod{N}$. This is the period of the sequence of powers of a .

Example 15.1.2. Show that the order of x satisfies $r \leq N$.

As $\gcd(x, N) = 1$, from Euler's formula $x^{\varphi(N)} \equiv 1 \pmod{N}$. $\varphi(N)$ is the number of y such that $\gcd(y, N) = 1$ and $y < N$, hence $\varphi(N) < N$. Therefore, there always exists a number $r \leq N$, such that $x^r \equiv 1 \pmod{N}$.

Some additional notation will be helpful for discussing the Order Finding Problem. If a, b and N are integers with $N \geq 1$ we write

$$a \equiv b \pmod{N}$$

to mean $N|(a - b)$ (shorthand for N divides $a - b$). As I mentioned before, we let \mathbb{Z}_N denote the set $\mathbb{Z}_N = \{0, \dots, N-1\}$. When we also associate with \mathbb{Z}_N the operations of addition and multiplication modulo N , forms a ring. If in addition N is prime, then \mathbb{Z}_N forms a field. We write \mathbb{Z}_N^* to denote the following set:

$$\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$$

The number of elements in \mathbb{Z}_N^* determines a function called the Euler φ -function: $\varphi(N) = |\mathbb{Z}_N^*|$. When we associate with the set \mathbb{Z}_N^* the operation of multiplication modulo N , it forms a group. This implies that for any element $a \in \mathbb{Z}_N^*$, there exists a unique element $b \in \mathbb{Z}_N^*$ that satisfies

$$ab \equiv 1 \pmod{N}$$

We generally write $a^{-1} \pmod{N}$ (or just a^{-1} when N is understood) to denote this element b .

Now, we can define the order of a given element $a \in \mathbb{Z}_N^*$, which is what the order-finding problem concerns. For $a \in \mathbb{Z}_N^*$, the order of a in \mathbb{Z}_N^* (or the order of a modulo N) is the smallest positive integer r such that

$$a^r \equiv 1 \pmod{N}$$

The fact that every $a \in \mathbb{Z}_N^*$ indeed has a well-defined order follows from Euler's theorem, which states that

$$a^{\varphi(N)} \equiv 1 \pmod{N}$$

for any integers a and $N \geq 2$ with $\gcd(a, N) = 1$. It turns out that the order of any element $a \in \mathbb{Z}_N^*$ is always a divisor of $\phi(N)$.

For example, let $a = 4$ and $N = 35$. because $\gcd(4, 35) = 1$ we have $4 \in \mathbb{Z}_{35}^*$. Computing powers of 4 modulo 35 gives:

$$4^1 \equiv 4, \quad 4^2 \equiv 16, \quad 4^3 \equiv 29, \quad 4^4 \equiv 11, \quad 4^5 \equiv 9, \quad 4^6 \equiv 1$$

(where all congruences are of course modulo 35). Thus, the order of 4 modulo 35 is 6.

Now that we know how the order of a given element $a \in \mathbb{Z}_N^*$ is defined, we can state the order finding problem.

15.2 The Problem Definition

Definition 15.2.1. Input: A positive integer $N \geq 2$ and an element $a \in \mathbb{Z}_N^*$

Output: The order of a in \mathbb{Z}_N^*

The input to the order finding problem is two n -bit integers m and a , where $m \geq 2$ and $a \in \mathbb{Z}_N^*$. The output is $\text{ord}_N(a)$, the order of a modulo N .

Example 15.2.1. Show that the order of a satisfies $r = \text{ord}_N(a) \leq N$.

15.3 The Classical Solution

Classically this problem is hard (at least as far as we know). Certainly the obvious approach of computing powers of a modulo N until 1 is obtained can take time exponential in $\log N$. A brute-force algorithm is to compute the sequence of powers of a (a, a^2, a^3, \dots) until $a \equiv 1$ is reached. That can take exponential-time when the modulus is an n -bit number because the order can be exponential in n (where n is the number of bits), thus exponential in number of bits. Although each power of a can be computed efficiently by the repeated-squaring trick, but there are potentially many different powers of a to check. In fact, there is no known algorithm to solve the problem using resources polynomial in the $\mathcal{O}(\lceil \log N \rceil)$ bits needed to specify the problem, where $\lceil \log N \rceil$ is the number of bits needed to specify N .

But why should we care about solving the order-finding problem efficiently? It might come across as an esoteric problem in computational number theory. One reason to care is that the factoring problem reduces to the order-finding problem. If we can solve the order-finding problem efficiently then we can use that to factor numbers efficiently.

What I'm going to do next is show how any efficient algorithm (classical or quantum) for the order-finding problem can be turned into an efficient algorithm for factoring. This is true for the general factoring problem; however, for simplicity, I'm going to explain it for the case of factoring numbers that are the product of two distinct primes. That's the hardest case, and the case that occurs in cryptographic applications.

Let our input to the factoring problem be an n -bit number m , such that $m = pq$, where p and q are distinct primes. Our goal is to determine p and q with a gate-cost that is polynomial in n .

The first step is to select an $a \in \mathbb{Z}_m^*$ and use our quantum order-finding algorithm (in the next section we will see that there is an efficient algorithm for the order-finding problem that we can use for this) as a subroutine to compute $r = \text{ord}_m(a)$. Note that, since $a^r \equiv 1 \pmod{m}$, it holds that $a^r - 1$ is a multiple of m . in other words, m divides $a^r - 1$.

Now, the order r is either an even or an odd number (it can go either way, depending on which a we pick). something interesting happens when r is an even number. If r is even then a^r is a square, with square root $a^{r/2}$, and we can express $a^r - 1$ using the standard formula for a difference of squares as

$$a^r - 1 = (a^{r/2} + 1)(a^{r/2} - 1)$$

Since m divides $a^r - 1$ and $m = pq$, it follows that p and q each divide $a^r - 1$. It's well known that if a prime divides a product of two numbers then it must divide one of them. Also, it's not possible that both p and q divide the factor $a^{r/2} - 1$. Why not? Because that would contradict the fact that r is, by definition, the smallest number for which $a^r \equiv 1 \pmod{m}$. Therefore, there are three possibilities for the factors that p and q divide:

1. p divides $a^{r/2} + 1$ and q divides $a^{r/2} - 1$.
2. q divides $a^{r/2} + 1$ and p divides $a^{r/2} - 1$.
3. p and q both divide $a^{r/2} + 1$.

In the fist two case p and q divide different factors; in the third case, they divide the same factor. Our reduction works well when r is even and p and q divide different factors. With this in mind, let's define a to be "lucky" when these conditions occurs.

Definition 15.3.1. (of a lucky $a \in \mathbb{Z}_m^*$). With respect to some $m \geq 2$, define $a \in \mathbb{Z}_m^*$ to be lucky if $\text{ord}_m(a)$ is an even number and m does not divide $a^{r/2} + 1$.

Given $a \in \mathbb{Z}_m^*$ that is lucky in the above sense, it holds that

$$\gcd(a^{r/2} + 1, m) \in \{p, q\}$$

This enables us to easily factor m by computing $\gcd(a^{r/2} + 1, m)$. The repeated squaring trick enables us to compute $a^{r/2}$, with a gate cost of $\mathcal{O}(n^2 \log n)$.

The outstanding matter is to find a luck $a \in \mathbb{Z}_m^*$. How do we do that? Unfortunately, there is no deterministic method known for finding a lucky $a \in \mathbb{Z}_m^*$. However, it's known that, for any $m \geq 2$, the number of lucky $a \in \mathbb{Z}_m^*$ is quite large.

Lemma 15.3.1. *For all $m = pq$, where p and q are distinct primes, at least half of the elements of \mathbb{Z}_m^* are lucky.*

So what we can do is randomly select an $a \in \mathbb{Z}_m^*$. The selection will be lucky with probability at least $\frac{1}{2}$, and therefore the resulting procedure successfully factors m with probability at least $\frac{1}{2}$. We can repeat this process to boost the success probability.

So that's how an efficient algorithm for order-finding problem can be used to factor a number efficiently. Now we can focus our attention on finding an efficient algorithm for order-finding.

15.4 The Quantum Solution

Let's begin with an input instance to the order-finding problem, N and a , which are both n -bit numbers and for which $a \in \mathbb{Z}_N^*$. The goal of the algorithm is to determine $r = \text{ord}_N(a)$.

Our main goal for the remainder of the chapter will be to show that the order finding problem can be solved using the method of phase estimation. Recall that for this framework, we need a unitary operation and an eigenvector.

Assume $N \geq 2$ is given and let n be the number of bits needed to encode elements of \mathbb{Z}_N in binary (so $n = \lfloor \log_2(N - 1) \rfloor + 1 = \lceil \log_2 N \rceil$). Given any $a \in \mathbb{Z}_N^*$, define an n -qubit transformation M_a as

$$M_a |x\rangle = |ax \pmod{N}\rangle$$

for every $x \in \mathbb{Z}_N$. This is not a complete specification of M_a because it does not specify $M_a |x\rangle$ for $N \leq x < 2^n$, but we will not care about its action on such states.

For the sake of choosing a well-defined transformation, we may say for simplicity that $M_a |x\rangle = |x\rangle$ for $N \leq x < 2^n$. The transformation M_a is reversible and therefore unitary. This follows from the fact that it maps classical states to classical states, along with the observation that it has an inverse: $M_{a^{-1}} M_a = I$, where a^{-1} is the inverse of a modulo N .

Example 15.4.1. Show that M_a is unitary. Consider the following matrix for the unitary transformation as

$$\begin{pmatrix} \tilde{M}_a & 0 \\ 0 & I \end{pmatrix}$$

Thus, it suffices to show that \tilde{M}_a is Unitary. Note that $0 \leq y, y' \leq N - 1$. Thus,

$$\langle y' | \tilde{M}_a^\dagger U | y \rangle = \langle xy' | xy \rangle$$

If $xy' = xy \pmod{N} \implies y' = y \pmod{N}$ because x is co-prime to N . Since, $0 \leq y \leq N - 1$ Thus $y' = y \pmod{N} \implies y = y'$. Thus,

$$\langle y' | \tilde{M}_a^\dagger \tilde{M}_a | y \rangle = \langle xy' | xy \rangle = \langle y | y' \rangle = \delta_{y,y'}$$

Hence, \tilde{M}_a is unitary.

There are two important requirements for us to be able to use the phase estimation procedure: we must have efficient procedures to implement a controlled- U^{2^j} operation for an integer j , and we must be able to efficiently prepare an eigen state $|\psi\rangle$ with a non-trivial eigenvalue, or at least a superposition of such eigenstates. Let us consider subjecting the transformation M_a to phase estimation. It will turn out that in doing this we will be able to determine the order of a modulo N . One can efficiently implement a reversible circuit for performing M_a given that the functions $f(x) = ax \pmod{N}$ and $g(x) = a^{-1}x \pmod{N}$ are efficiently computable by Boolean circuits. If we wish to subject this transformation to phase estimation, however, recall that we will need to have an efficient implementation of $\Lambda_m(M_a)$ for m (roughly) corresponding to the number of bits of precision we need. In fact, this transformation can be expressed as

$$\Lambda_m(M_a) |k\rangle |x\rangle = |k\rangle |a^k x \pmod{N}\rangle$$

can also be implemented efficiently. Specifically, based on the modular exponentiation algorithm, it can be implemented using $O(mn^2)$ gates. We will need to wait and see how precise our procedure needs to be to determine the order of a , but we may keep in the back of our minds that $m = O(n)$ will be sufficient.

What are the eigenvectors and eigenvalues of M_a ? It turns out that there are lots, but we will only need to consider a subset of them. Letting r be the order of $a \in \mathbb{Z}_N^*$, we see that the following vector is an eigenvector of M_a :

$$|\psi_0\rangle = \frac{1}{\sqrt{r}}(|1\rangle + |a\rangle + |a^2\rangle + \dots + |a^{r-1}\rangle)$$

(From now on the operations inside kets are implicitly assumed to be modulo N). To see that it is an eigenvector, just compute:

$$\begin{aligned} M_a |\psi_0\rangle &= \frac{1}{\sqrt{r}}(|a\rangle + |a^2\rangle + |a^3\rangle + \dots + |a^r\rangle) \\ &= \frac{1}{\sqrt{r}}(|a\rangle + |a^2\rangle + \dots + |a^{r-1}\rangle + |1\rangle) \\ &= |\psi_0\rangle \end{aligned}$$

So, the associated eigenvalue is 1. Here is another.

$$|\psi_1\rangle = \frac{1}{\sqrt{r}}(|1\rangle + \omega_r^{-1}|a\rangle + \omega_r^{-2}|a^2\rangle + \dots + \omega_r^{-(r-1)}|a^{r-1}\rangle)$$

where as before $\omega_r = e^{2\pi i/r}$. We have

$$\begin{aligned} M_a |\psi_1\rangle &= \frac{1}{\sqrt{r}}(|a\rangle + \omega_r^{-1}|a^2\rangle + \omega_r^{-2}|a^3\rangle + \dots + \omega_r^{-(r-1)}|a^r\rangle) \\ &= \frac{\omega_r}{\sqrt{r}}(\omega_r^{-1}|a\rangle + \omega_r^{-2}|a^2\rangle + \omega_r^{-3}|a^3\rangle + \dots + \omega_r^{-r}|a^r\rangle) \\ &= \frac{\omega_r}{\sqrt{r}}(\omega_r^{-1}|a\rangle + \omega_r^{-2}|a^2\rangle + \omega_r^{-3}|a^3\rangle + \dots + |1\rangle) \\ &= \frac{\omega_r}{\sqrt{r}}(|1\rangle + \omega_r^{-1}|a\rangle + \omega_r^{-2}|a^2\rangle + \omega_r^{-3}|a^3\rangle + \dots + \omega_r^{-(r-1)}|a^{r-1}\rangle) \\ &= \omega_r |\psi_1\rangle \end{aligned}$$

In general, for

$$|\psi_j\rangle = \frac{1}{\sqrt{r}}(|1\rangle + \omega_r^{-j}|a\rangle + \omega_r^{-2j}|a^2\rangle + \dots + \omega_r^{-j(r-1)}|a^{r-1}\rangle)$$

we have

$$M_a |\psi_j\rangle = \omega_r^j |\psi_j\rangle$$

Therefore $|\psi_j\rangle$ is an eigenvector of M_a with eigenvalue $\omega_r^j = e^{2\pi ij/r}$. If we can estimate the eigenvalues's parameter j/r where j (where j is known somehow to us) with sufficiently many bits of precision, then we can deduce what r is. This requires us to efficiently simulate a multiplicity controlled $\Lambda_m(M_a)$ gate and also to construct the state $|\psi_j\rangle$.

15.4.1 Multiplicity-controlled $-U_{a,N}$ gate

This can be done using modular exponentiation, with which we can implement the entire sequence of controlled U^{2^j} operations applied by the phase estimation procedure using $\mathcal{O}(\lceil \log N \rceil)$ gates. Here I will show you a rough sketch of how to efficiently compute a multiplicity controlled-U gate. Consider the mapping of the multiplicity controlled-U gate. Consider the mapping of the multiplicity controlled $U_{a,N}$ gate with m control bits. For each $x \in \{0, 1\}^m \equiv \{0, 1, \dots, 2^m - 1\}$ and $b \in \mathbb{Z}_N^*$, this gate

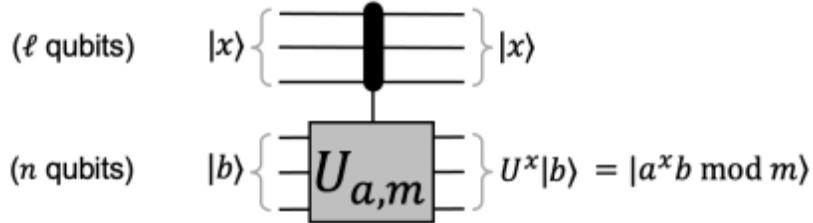


Figure 15.1: Multiplicity-controlled $U_{a,m}$ gate

maps $|x\rangle|b\rangle$ to

$$|x\rangle(U_{a,N})^x|b\rangle = |x\rangle|a^x b \bmod N\rangle$$

(which is equivalent to $U_{a,N}$ being applied x times).

We can compute this efficiently (with respect to m and n) by using the repeated squaring trick to exponentiate a . The number of gates is $O(mn \log n)$. However, please note that in general, if we can efficiently compute some unitary U then it does not always follow that we can compute a multiplicity-controlled- U gate efficiently. The $U_{a,N}$ that arises here has special structure that permit this.

Important Note

Modular exponentiation

How can we compute the sequence of controlled U^{2^j} operations used by the phase estimation procedure as part of the order-finding algorithm? that is, we wish to compute the transformation

$$\begin{aligned} |z\rangle|y\rangle &\rightarrow |z\rangle U^{z_t 2^{t-1}} \dots U^{z_1 2^0}|y\rangle \\ &= |z\rangle|x^{z_t 2^{t-1}} \times \dots \times x^{z_1 2^0}(\text{mod } N)\rangle \\ &= |z\rangle|x^z y(\text{mod } N)\rangle \end{aligned}$$

Thus the sequence of controlled- U^{2^j} operations used in phase estimation is equivalent to multiplying the contents of the second register by the modular exponential x^z (mod N), where z is the contents of the first register. The operation may be accomplished easily using the techniques of reversible computation. The basic idea is to reversibly compute the function x^z (mod N) of z in a third register, and then to reversibly multiply the contents of the second register by x^z (mod N), using the trick of uncomputation to erase the contents of the third register upon completion. The algorithm for computing the modular exponential has two stages. The first stage uses modular multiplication to compute x^2 (mod N), by squaring x modulo N , then computes x^4 (mod N) by squaring x^2 (mod N), and continues in this way, computing x^{2^j} (mod N) for all j up to $t - 1$. We use $t = 2n + 1 + \lceil \log_2(2 + \frac{1}{2\epsilon}) \rceil = \mathcal{O}(n)$, so a total of $t - 1 = \mathcal{O}(n)$ squaring operations is performed at a cost of $\mathcal{O}(n^2)$ each (this cost assumes the circuit used to do squaring implements the familiar algorithm we all learn as children for multiplication), for a total cost of $\mathcal{O}(n^3)$ for the first stage. The second stage of the algorithm is based upon the observation we've already noted,

$$x^z(\text{mod } N) = (x^{z_t 2^{t-1}}(\text{mod } N))(x^{z_{t-1} 2^{t-2}}(\text{mod } N)) \dots (x^{z_1 2^0}(\text{mod } N))$$

Performing $t - 1$ modular applications with a cost $\mathcal{O}(n^2)$ each, we see that this product can be computed using $\mathcal{O}(n^3)$ gates. This is sufficiently efficient for our purposes, but more efficient algorithms are possible based on more efficient algorithms for multiplication. Using the techniques, it is now straightforward to construct a reversible circuit with a t bit register and an n bit register which, when started in the state (z, y) outputs $(z, x^z y(\text{mod } N))$ using $\mathcal{O}(n^3)$ gates computing the transformation $|z\rangle|y\rangle \rightarrow |z\rangle|x^z y(\text{mod } N)\rangle$.

Example 15.4.2. Prove that

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \omega_r^{sk} |\psi_s\rangle = |a^k \bmod N\rangle$$

Recall that $|\psi_k\rangle = \sum_{j=0}^{r-1} \frac{1}{\sqrt{r}} \omega_r^{-jk} |a^j \bmod N\rangle$. Upon substitution we get,

$$\begin{aligned} \frac{1}{r} \sum_{s=0}^{r-1} \sum_{j=0}^{r-1} \omega_r^{(k-j)s} |a^j \bmod N\rangle &= \frac{1}{r} \sum_{j=0}^{r-1} \left(\sum_{s=0}^{r-1} \omega_r^{(k-j)s} \right) |a^j \bmod N\rangle \\ &= \frac{1}{r} \sum_{j=0}^{r-1} \left(\frac{\omega^{r(k-j)} - 1}{\omega^{k-j} - 1} \right) |a^j \bmod N\rangle \\ &= \frac{1}{r} \sum_{j=0}^{r-1} (r\delta_{k,j}) |a^j \bmod N\rangle \\ &= |a^k \bmod N\rangle \end{aligned}$$

Example 15.4.3. The quantum state is produced in the order-finding algorithm, before the inverse fourier transform, is

$$|\psi\rangle = \sum_{j=0}^{2^t-1} |j\rangle U^j |1\rangle = \sum_{j=0}^{2^t-1} |j\rangle |x^j \bmod N\rangle$$

if we initialize the second register as $|1\rangle$. Show that the same state is obtained if we replace U^j with a different unitary transform V , which computes

$$V|j\rangle|k\rangle = |j\rangle|k + x^j \bmod N\rangle$$

and start the second register in the state $|0\rangle$. Also how to construct V using $\mathcal{O}(n^3)$ gates.

For V ,

$$|\psi\rangle = \sum_{j=0}^{2^t-1} V|j\rangle|0\rangle = \sum_{j=0}^{2^t-1} |j\rangle|0 + x^j \bmod N\rangle$$

Thus, V does the same unitary transformation as U^j . Writing $x^j(\bmod N) = (x^{j_t 2^{t-1}}(\bmod N))(x^{j_{t-1} 2^{t-2}}(\bmod N)) \dots (x^{j_1 2^0}(\bmod N))$, each modular multiplication requires $\mathcal{O}(L^2)$ gates, hence for the total product of $t-1$ modular multiplications we require $\mathcal{O}(L^3)$ gates, and uses the circuit shown in figure. the addition of k is done after the modular multiplication and requires $\mathcal{O}(L)$ gates, hence in total we will require $\mathcal{O}(L^3)$ gates.

Example 15.4.4. Show that the least common multiple of positive integers x and y is $xy/\gcd(x, y)$, and thus may be computed in $\mathcal{O}(L^2)$ operations if x and y are L bit numbers.

The least common multiple (LCM) of two positive integers x and y is the smallest positive integer that is a multiple of both x and y . The greatest common divisor (GCD) of x and y is the largest positive integer that divides both x and y without leaving a remainder.

Let $m = \text{LCM}(x, y)$ be the lowest common multiple. Let M be any common multiple. Then we can write $M = mq + r$. x and y divide both M and m , hence they also divide r , meaning it's a common multiple, but $r < m$ and m is the lowest common multiple, therefore $r = 0$. Now let $x = \gcd(x, y)x_1$ and $y = \gcd(x, y)y_1$ with $\gcd(x_1, y_1) = 1$. x and y divide $\gcd(x, y)x_1y_1$ hence it's a common multiple, therefore we can write $\gcd(x, y)x_1y_1 = mq_1$. Therefore, we have $x_1 = \frac{m}{y}q_1$ and $y_1 = \frac{m}{x}q_1$, hence q_1 divides both x_1 and y_1 . However, $\gcd(x_1, y_1) = 1$, hence $q_1 = 1$. Hence, $\text{LCM}(x, y) = \gcd(x, y)x_1y_1 = \gcd(x, y)x_1\gcd(x, y)y_1/\gcd(x, y) = xy/\gcd(x, y)$. We can use Stein's gcd algorithm which requires $\mathcal{O}(L^2)$ gates.

We glossed over some technical details about how the multiplicity controlled $U_{a,N}$ gate can be implemented. That gate is a unitary operation such that

$$|x, b\rangle \rightarrow |x, a^x b\rangle$$

for all $x \in \{0, 1\}^m$ and $b \in \mathbb{Z}_N^*$. But our framework for computing classical functions in superposition is different from this. What we showed there is that we can compute an f -query

$$|x, c\rangle \rightarrow |x\rangle |f(x) \oplus c\rangle$$

in terms of a classical implementation of f .

There are two remedies. One is a separate construction for efficiently computing the mapping. Such an efficient construction exists; however, I will not explain it here. instead, I will show how to use a mapping of the form as follows:

Define $f_{a,N} : \{0, 1\}^m \rightarrow \{0, 1\}^n$ as $f_{a,N}(x) = a^x$. Then, since there is a classical algorithm for efficiently computing $a^x b \bmod m$ in terms of (x, a, b) we can efficiently compute the mapping $|x, c\rangle \rightarrow |x, f_{a,N}(x) \oplus c\rangle$. And then we can use the circuit in figure 15.1 instead of the circuit in figure 15.2. The outputs of the two circuits are the same because, for both circuits, the state right after the multiplicity -controlled $U_{a,N}$ gate and $f_{a,N}$ query is

$$\frac{1}{\sqrt{2^m}} \sum_{x \in \{0,1\}^m} |x\rangle |a^x\rangle$$

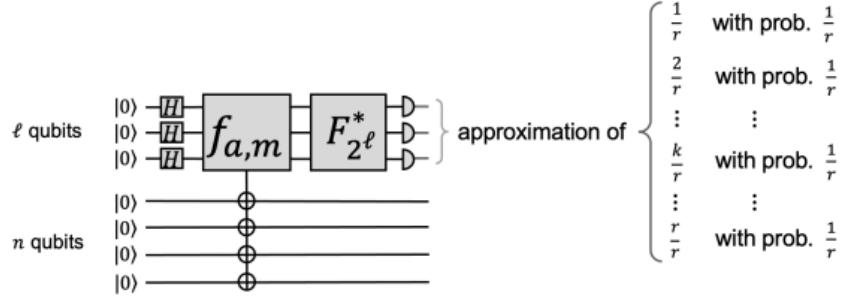
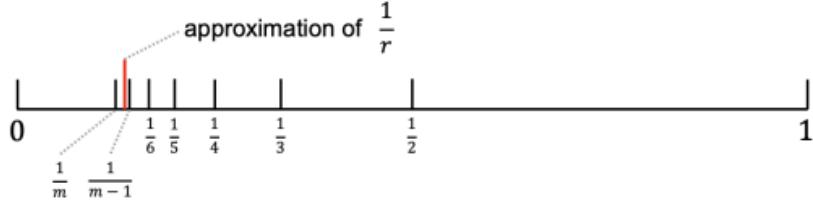


Figure 15.2: Multiplicity-Controlled U operation

15.4.2 Precision needed to determine $1/r$

Now, how many bits of precision m should we use in our phase estimation of $1/r$? It should be sufficient to uniquely determine $\frac{1}{r}$. We know that $r \in \{1, 2, 3, \dots, N\}$ and the corresponding reciprocals are $\{\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \dots, \frac{1}{N}\}$. Here are the positions of these reciprocals on the real line. The smallest gap occurs between $\frac{1}{N}$ and $\frac{1}{N-1}$. The size

Figure 15.3: The positions of $\{\frac{1}{1}, \frac{1}{2}, \dots, \frac{1}{m}\}$ on the real line.

of this gap is

$$\frac{1}{N-1} - \frac{1}{N} = \frac{N - (N-1)}{N(N-1)} = \frac{1}{N(N-1)} \approx \frac{1}{N^2}$$

This means the approximation must be within $\frac{1}{2N^2}$ of $\frac{1}{r}$. Since N is an n -bit number $N \approx 2^n$ and $\frac{1}{2N^2} \approx \frac{1}{2^{2n+1}}$ which corresponds to setting $m = 2n$.

One another way to think about this is as follows. Remember that our goal is to find r . If we have some integer $j \in \{0, \dots, 2^m - 1\}$ for which

$$\frac{j}{2^m} \approx \frac{1}{r}$$

there is an obvious strategy that (hopefully) will find r . In other words, whether it is with a calculator or (more likely) with an ordinary classical computer, compute the

reciprocal of $j/2^m$. Although r must be an integer, you probably would not get an integer when you compute $2^m/j$ because $j/2^m$ is only an approximation to r . The natural thing to do is round off to the nearest integer, so your guess for r would be

$$\lfloor \frac{2^m}{j} + \frac{1}{2} \rfloor$$

Now, if your approximation $j/2^l$ to $1/r$ does not have sufficiently many bits of precision, you cannot be sure your answer is correct. So how accurately do we need to approximate $1/r$ to be correct? Intuitively, you need enough precision to discriminate between estimates for $1/(r-1)$, $1/r$, $1/(r+1)$, and so on, so a good guess is that the approximation from the phase estimation procedure should be within $1/(2r(r+1))$ of the correct value - because this is half the smallest distance between $1/r$ and $1/s$ for some integer $s \neq r$. Of course we do not know what r is, but we do know that $r < N$. So let us guess that it is sufficient that our approximation satisfies

$$\frac{j}{2^m} = \frac{1}{r} - \epsilon$$

for ϵ satisfying

$$|\epsilon| \leq \frac{1}{2N^2}$$

Indeed this accuracy is sufficient. We can be sure that rounding $2^m/j$ to the nearest integer will give r is

$$\left| \frac{2^m}{j} - r \right| < \frac{1}{2}$$

Assuming that $|\epsilon| \leq 1/(2N^2)$ implies

$$\left| \frac{2^m}{j} - r \right| = \left| \frac{1}{\frac{1}{r} - \epsilon} - r \right| = \left| \frac{r^2 \epsilon}{1 - r\epsilon} \right| \leq \frac{\frac{r^2}{2N^2}}{1 - \frac{r}{2N^2}} = \frac{r^2}{2N^2 - r} \leq \frac{(N-1)^2}{2N^2 - N} \leq \frac{1}{2}$$

Therefore, if we take $m = 2n$ in the phase estimation procedure, the resulting accuracy will be sufficient to find r .

15.4.3 Can we construct $|\psi_1\rangle$?

To efficiently implement the phase estimation algorithm, we also need to be able to effectively create the eigenvector $|\psi_1\rangle$. Of course, if we already know what r is, then this is straightforward. But the algorithm not know r at this stage; r is what the algorithm is trying to determine. It seems very hard to construct this state without

knowing what r is. This is a serious problem; it's now known how to construct this state directly. Instead of producing that state $|\psi_1\rangle$, our way forward will be to use a different state.

At this point, we don't know how to get our hands on any of these eigenvectors, but let's imagine, for the sake of understanding how phase estimation could help us with our problem, that we have a copy of the state $|\psi_1\rangle$. Consider the phase estimation procedure for this eigenvector as shown in figure 15.4.

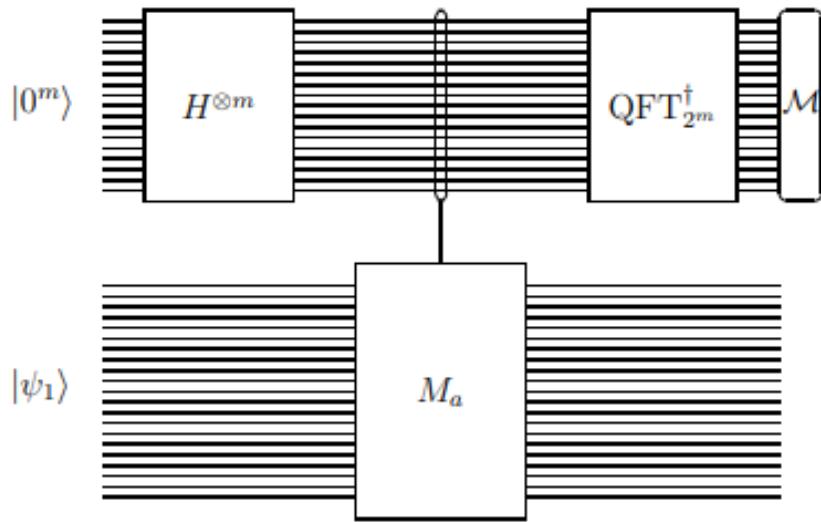


Figure 15.4: Order Finding Circuit with $|\psi_1\rangle$ as eigenvector

The eigenvalue associated with $|\psi_1\rangle$ is $\omega_r = e^{2\pi i(1/r)}$. Assuming we were to perform the procedure several times and take the most commonly appearing approximation $j/2^m$ that, with very high probability, is within distance $1/2^{m+1}$ to $1/r$. I mentioned this before, but it is worth noting again that we only need one copy of the eigenvector $|\psi_1\rangle$ even if we want to run the phase estimation procedure several times, because the eigenvector is unaffected by the phase estimation procedure.

15.4.4 Order-finding using a random eigenvector $|\psi_k\rangle$

We still have a big problem to contend with, however, which is that we do not know how to get our hands on $|\psi_1\rangle$. Obtaining this vector is probably no easier than finding r , so we will have to change our approach slightly. Let's consider alternatives

to the eigenvector $|\psi_1\rangle = \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} \omega^{-j} |a^j\rangle$ (with $\omega = e^{2\pi i \frac{1}{r}}$). Other eigenvectors are

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{r} \sum_{j=0}^{r-1} \omega^{-2j} |a^j\rangle \\ |\psi_k\rangle &= \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} \omega^{-kj} |a^j\rangle \\ &\vdots \\ |\psi_r\rangle &= \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} \omega^{-rj} |a^j\rangle \end{aligned}$$

for each $k \in \{0, 1, 2, \dots, r-1\}$, the eigenvalue of $|\psi_k\rangle$ is $e^{2\pi i \frac{k}{r}}$.

Suppose that we are able to devise an efficient procedure that somehow produces a random sample from the r different eigenvectors (where each $|\psi_k\rangle$) occurs with probability $\frac{1}{r}$). Can we deduce r from this? There are two versions of this scenario depending on whether or not the procedure reveals k .

When k is known

First, let's suppose that the output of the procedure is the pair $(k, |\psi_k\rangle)$, with each possibility occurring with probability $\frac{1}{r}$. In this case, we can use the phase estimation algorithm (with random $|\psi_k\rangle$) to get an approximation of $\frac{k}{r}$ and then divide the approximation by k to turn it into an approximation of $\frac{1}{r}$, and proceed from there as with the case of $|\psi_1\rangle$. The details of this case are straightforward.

When k is not known

Now, let's change the scenario slightly and assume that we have an efficient procedure that somehow generates a random $|\psi_k\rangle$ (uniformly distributed among the r possibilities) as output - but without revealing k . Can we still extract r from $|\psi_k\rangle$ alone?

Using the phase estimation algorithm, we can obtain a binary fraction $0.b_1b_2\dots b_m$ that estimates $\frac{k}{r}$ within m -bits of precision (where we can set the value of m). but how can we determine k and r from this? This is a tricky problem, because we don't know what k to divide by in order to turn an approximation of $\frac{k}{r}$ into an approximation of $\frac{1}{r}$. But there is a way to do this using the fact that we have an upper bound N on the denominator.

Let's carefully restate the situation as follows. We have an n -bit integer N and we are also given an m -bit approximation $0.b_1b_1\dots b_m$ of one of the elements of

$$\left\{ \frac{k}{r} \mid \text{where } r \in |\{1, 2, \dots, m\} \text{ and } k \in \{0, 1, 2, \dots, r-1\}\right\}$$

Our goal is to determine which element of the set is being approximated.

Consider the example where $N = 8$ (so the maximum denominator is N). The following figure 15.5 shows all the candidates for $\frac{k}{r}$. Note that denominators 2, 3

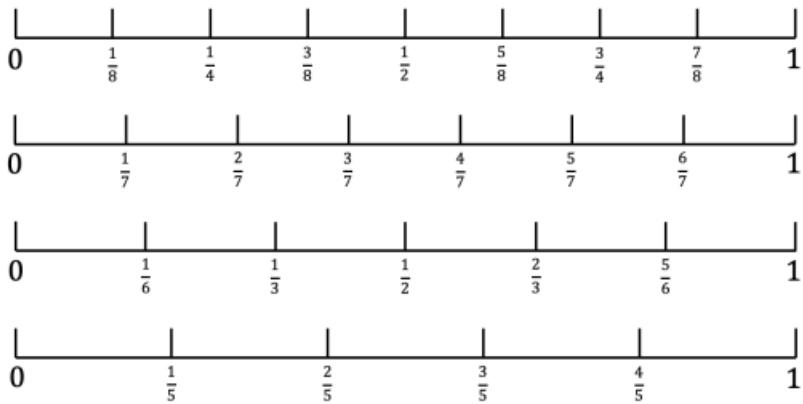


Figure 15.5: The set of candidates for $\frac{k}{r}$ in the $m = 8$ case

and 4 are covered by the cases of 6 and 8 in the denominator. Figure 15.6 shows what all these candidates look like when they're combined on one line. Now suppose

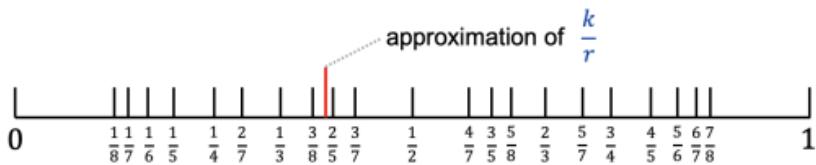


Figure 15.6: The set of candidates for $\frac{k}{r}$ in the $m = 8$ case (shown combined)

that we are able to obtain a 7-bit approximation to one of these candidates. Say it's 0.0110011. This number is exactly $\frac{51}{128}$, but that's not one of the candidates. The rational number $\frac{2}{5}$ is the unique candidate within distance of $\frac{1}{2^7}$ from 0.0110011

In general, how good an approximation do we need in order to single out a unique $\frac{k}{r}$? this depends on the smallest gap among the set of candidates. It turns out that

the gap is smallest at the ends: $\frac{1}{N-1} - \frac{1}{N} \approx \frac{1}{N^2}$. Therefore, setting $m = 2n$ provides the correct level of precision to guarantee that there is a unique rational number $\frac{k}{r}$ that's close to the approximation.

But how do we find the rational number? If N is an n -bit number then there are exponentially many candidates to search among. So a brute-force is not going to be efficient.

It turns out that there is a known efficient algorithm that's tailor made for this problem called the continued fractions algorithm. The continued-fractions algorithm enables us to determine k and r efficiently from a m -bit approximation of $\frac{k}{r}$.

But perhaps you've already noticed that there is a major problem will all this: that of reduced fractions.

Continued fractions

The reduction of order-finding to phase estimation is completed by describing how to obtain the desired answer, r , from the result of the phase estimation algorithm $\phi \approx k/r$. We only know ϕ to $2n + 1$ bits, but we also know a priori that it is a rational number - the ratio of two bounded integers and if we could compute the nearest such fraction to ϕ we might obtain r .

There is an algorithm which accomplishes this task efficiently, known as the continued fractions algorithm. The reason this algorithm satisfies our needs is the following theorem.

Theorem 15.4.1. *Suppose $\frac{k}{r}$ is a rational number such that*

$$\left| \frac{k}{r} - \phi \right| \leq \frac{1}{2r^2}$$

Then $\frac{k}{r}$ is a convergent of the continued fraction for ϕ , and thus can be computed in $\mathcal{O}(n^3)$ operations using the continued fractions algorithm.

Important Note

The continued fractions algorithm

The idea of the continued fractions algorithms is to describe real numbers in terms of integers alone, using expressions of the form

$$[a_0, \dots, a_M] = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\dots + \frac{1}{a_M}}}}$$

where a_0, \dots, a_M are positive integers. (For applications to quantum computing it is convenient to allow $a_0 = 0$ as well.) We define m th convergent ($0 \leq m \leq M$) to this continued fraction to be $[a_0, \dots, a_m]$. The continued fractions algorithm is a method for determining the continued fraction expansion of an arbitrary real number. It is easily understood by example. Suppose we are trying to decompose $31/13$ as a continued fraction. The first step of the continued fractions algorithm is to split $31/13$ into its integer and fractional part,

$$\frac{31}{13} = 2 + \frac{5}{13}$$

next we invert the fractional part, obtaining

$$\frac{31}{13} = 2 + \frac{1}{\frac{13}{5}}$$

These steps - split then invert - are not applied to $13/5$, giving

$$\frac{31}{13} = 2 + \frac{1}{2 + \frac{3}{5}} = 2 + \frac{1}{2 + \frac{1}{\frac{5}{3}}}$$

Next we split and invert $5/3$:

$$\frac{31}{13} = 2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{1 + \frac{2}{3}}}} = 2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{1 + \frac{1}{2}}}}$$

The decomposition into a continued fraction now terminates since

$$\frac{3}{2} = 1 + \frac{1}{2}$$

may be written with a 1 in the numerator without any need to invert, giving a final continued fraction representation of $31/13$ as

$$\frac{31}{13} = 2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{2}}}}}$$

It's clear that the continued fraction algorithm terminates after a finite number of 'split' and 'invert' steps for any rational number, since the numerators which appear (31,5,3,2,1 in the example) are strictly decreasing. How quickly does this termination occur? It turns out that if s/r is a rational number, and s and r are n bit integers, then the continued fraction expansion for ϕ can be computed using $\mathcal{O}(n^3)$ operations - $\mathcal{O}(n)$ 'split' and 'invert' steps, each using $\mathcal{O}(n^2)$ gates for elementary arithmetic.

Since ϕ is an approximation of k/r accurate to $2n + 1$ bits, it follows that $|k/r - \phi| \leq 2^{-2n-1} \leq 1/2r^2$, since $r \leq N \leq 2^n$. Thus, the theorem applies.

Summarizing, given ϕ the continued fractions algorithm efficiently produces numbers s' and r' with no common factor, such that $s'/r' = s/r$. The number r' is our candidate for the order. We can check whether it is the order by calculating $x^{r'}$ mod N , and seeing if the result is 1. If so, then r' is the order of a^r modulo N , and we are done!

The problem of reduced fractions is that different k and r can correspond to exactly the same number. For example, if $k = 34$ and $r = 51$ then $\frac{k}{r} = \frac{34}{51} = \frac{2}{3}$. There is no way to distinguish between $\frac{34}{51}$ and $\frac{2}{3}$. The continued fractions algorithm only provides a k and r in reduced form, which does not necessarily correspond to the actual r . If it happens that $\gcd(k, r) = 1$ then this problem does not occur. In that case, the fraction is already in the reduced form.

Recall that, in our setting we are assuming that k is uniformly sampled from the set $\{0, 1, 2, \dots, r - 1\}$. What can we say about the probability of such a random k being relatively prime to r ? The probability is the ratio of the size of the set \mathbb{Z}_r^* to the set \mathbb{Z}_r . The ratio need not be constant, but is too small. It is known to be at least $\Omega(1/\log \log r)$ in size. When the modulus is n -bits, this is at least $\Omega(1/\log n)$.

This means that $\mathcal{O}(\log n)$ repetitions of the process is sufficient for the probability of a k that's relatively prime to r to arise with constant probability. Procedurally, in the case where $\gcd(k, r) > 1$, the result is an r that's smaller than the order (which can be detected because in such cases $a^r \bmod m \neq 1$). The $\mathcal{O}(\log n)$ repetitions just introduces an extra factor into the gate cost.

In fact, we can do better than that. If we make two repetitions then there are two rational numbers $\frac{k_1}{r}$ and $\frac{k_2}{r}$ (where r is the order, which is unknown to us). Call the reduced fractions that we get from the continued fractions algorithm $\frac{k'_1}{r'_1}$ and $\frac{k'_2}{r'_2}$ (where r is a multiple of r'_1 and r'_2). It turns out that, with constant probability, the least common multiple of r'_1 and r'_2 is r . So, with just two repetitions of the phase estimation algorithm, we can obtain the order r with constant success probability (independent of n), assuming we use an independent random eigenvector in each run.

15.4.5 Conclusions of order-finding with a random eigenvector

Now, let's step back and see where we are. We're trying to solve the order finding problem using the phase-estimation algorithm.

We have a multiplicity controlled $U_{a,N}$ gate that's straightforward to compute efficiently.

Regarding the eigenvector, if we could construct the eigenvector $|\psi_1\rangle$ then we could determine the order r from that. Unfortunately, we don't know how to generate the state $|\psi_1\rangle$ efficiently. We also saw that: if we could generate a randomly sampled pair $(k, |\psi_k\rangle)$ then we could also determine r from that. unfortunately, we don't know how to generate such a random pair efficiently.

Next, we saw that: if we could generate a randomly sampled $|\psi_k\rangle$ (without the k) then we could still determine r from that. in that case, we had to do considerably more work. But, using the continued fractions algorithm and some probabilistic analysis, we could make this work. So can we generate a random $|\psi_k\rangle$? Unfortunately, we don't know how to generate such a random pair efficiently.

Are we at a dead end? No, in fact we're almost at a solution! What I'll show next is that if, instead of generating a random $|\psi_k\rangle$ (with probability $\frac{1}{r}$ for each k), we can instead generate a superposition of the $|\psi_k\rangle$ (with amplitude $\frac{1}{\sqrt{r}}$ for each k) then we can determine r from this. And this is a state that we can generate efficiently.

15.4.6 Order-finding using a superposition of eigenvectors

Remember the phase estimation algorithm at the very end it was discussed what happens if the target register is in a superposition of eigenvectors. In that case, the outcome is an approximation of the phase of a random eigenvector of that superposition, with probability of the amplitude squared. Therefore, setting the target register to state

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\psi_k\rangle$$

results in the same outcome that occurs when we use a randomly generated eigenvector state - which is the case that we previously analyzed. So we have an efficient algorithm for order-finding using the superposition state in place of an eigenvector (it succeeds with constant probability, independent of n).

Order finding without the requirement of an eigenvector

How can we efficiently generate the superposition state? In fact, this is extremely easy to do. To see how, expand this state as

$$\begin{aligned}
 \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\psi_k\rangle &= \frac{1}{r} (|1\rangle + |a\rangle + |a^2\rangle + \dots + |a^{r-1}\rangle) \\
 &\quad + \frac{1}{r} (|1\rangle + \omega|a\rangle + \omega^2|a^2\rangle + \dots + \omega^{r-1}|a^{r-1}\rangle) \\
 &\quad + \frac{1}{r} (|1\rangle + \omega^2|a\rangle + \omega^4|a^2\rangle + \dots + \omega^{2(r-1)}|a^{r-1}\rangle) \\
 &\quad + \frac{1}{r} (|1\rangle + \omega^3|a\rangle + \omega^6|a^2\rangle + \dots + \omega^{3(r-1)}|a^{r-1}\rangle) \\
 &\quad \vdots \\
 &\quad + \frac{1}{r} (|1\rangle + \omega^{r-1}|a\rangle + \omega^{2(r-1)}|a^2\rangle + \dots + \omega^{(r-1)(r-1)}|a^{r-1}\rangle) \\
 &= |1\rangle
 \end{aligned}$$

where the simplification to $|1\rangle$ is a consequence of $\omega^r = 1$ and the fact that, for all $k \in \{0, 1, 2, \dots, r-1\}$, it holds that $1 + \omega^k + \omega^{2k} + \dots + \omega^{(r-1)k} = \frac{\omega^{kr}-1}{\omega^k-1} = 0$, using the formula for summation of a geometric series since $\omega^k r = e^{2\pi i k r / r} = e^{2\pi i k} = 1$.

So we're left with $|1\rangle$. The n -bit binary representation of the number 1 is 00...01, so the superposition of the eigenvectors is merely the computational basis state $|00\dots01\rangle$, which is indeed trivial to construct. The quantum part of the order finding algorithm looks like this. The accuracy parameter m is set to $2n$. The out-

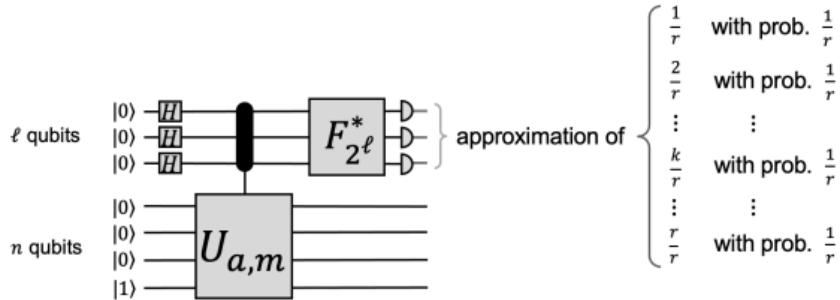


Figure 15.7: Circuit for Order Finding

put of the measurement is then post-processed by the classical continued fractions

algorithm, which produces a numerator k and denominator r . After two runs, taking the least common multiple of the two denominators, we obtain r , with constant probability. This constant probability is based on the phase estimation algorithm succeeding, in addition to r occurring via the continued fractions algorithm. The total gate cost is $\mathcal{O}(n^2 \log n)$.

The solution will be to run the phase estimation procedure on the state $|1\rangle$ rather than on an eigenvector. It follows from the observation that

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\psi_k\rangle = \frac{1}{r} \sum_{k=0}^{r-1} \sum_{l=0}^{r-1} \omega_r^{-kl} |a^l\rangle = |a^0\rangle = |1\rangle$$

that running the phase estimation procedure on the state $|1\rangle$ is equivalent to running the procedure on an eigenvector $|\psi_k\rangle$ for $k \in \{0, 1, \dots, r-1\}$ chosen uniformly at random. It is not obvious that this is so - it depends on the fact that the phase estimation procedure leaves eigenvectors unchanged, and that these eigenvectors form an orthonormal set. Specifically, if we were to run the phase estimation procedure on the state.

$$|1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\psi_k\rangle$$

the state immediately before the measurement would have the form

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\phi_k\rangle |\psi_k\rangle$$

where each $|\phi_k\rangle$ is the state of the first m qubits that you would get by running the phase estimation procedure just on $|\psi_k\rangle$. Because the state $|\psi_0\rangle, \dots, |\psi_{r-1}\rangle$ are orthonormal, the probability to obtain some value j from the measurement is just the average over $k \in \{0, \dots, r-1\}$ chosen uniformly to have measured that value starting with the eigenvector $|\psi_k\rangle$.

So, when we run the phase estimation procedure on $|1\rangle$, we may as well imagine that we instead ran the phase estimation procedure on an eigenvector $|\psi_k\rangle$ for $k \in \{0, \dots, r-1\}$ chosen uniformly at random. This will be almost as good as having $|\psi_1\rangle$. In other words, the phase estimation procedure would give an approximation

$$\frac{j}{2^m} \approx \frac{k}{r}$$

for $k \in \{0, \dots, r-1\}$ chosen uniformly. the question we must now address: if you have enough precision can you find r from such an approximation? It is important

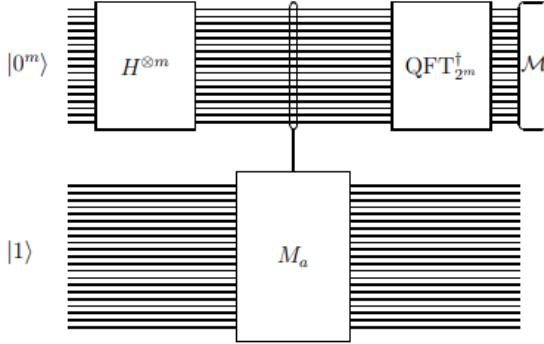


Figure 15.8: Order Finding Circuit with $|1\rangle$ as eigenvector

for us to keep in mind that k will not be known explicitly when we attempt to find r . The answer is that you cannot be guaranteed to find r given just one sample, but if you repeat the process several times (each time for a different k) you can find r with high probability. The way to do it is to use the continued fractions algorithm. The continued fractions can be stated as follows:

Fact: Given a real number $\alpha \in (0, 1)$ and $N \geq 2$, there is at most one fraction

$$\frac{x}{y}$$

with $0 \leq x, y < N$, $y \neq 0$, $\gcd(x, y) = 1$ and

$$\left| \alpha - \frac{x}{y} \right| < \frac{1}{2N^2}$$

Given α and N , the continued fraction will find x and y (if they exist) in $\mathcal{O}((\log N)^3)$ bit operations.

15.5 Performance

How can the order-finding algorithm fail? There are two possibilities. First, the phase estimation procedure might produce a bad estimate to s/r . This occurs with probability at most ϵ , and can be made small with a negligible increase in the size of the circuit. More seriously, it might be that s and r have a common factor, in which case the number r' returned by the continued fractions algorithm be a factor of r , and not r itself. Fortunately there are at least three ways around this problem.

Perhaps the most straightforward way is to note that for randomly chosen s in the range 0 to $r - 1$, it's actually pretty like that s and r are co-prime, in which case the continued fractions algorithm must return r . To see that this is the case, note that the number of prime numbers less than r is at least $r/2 \log r$, and thus the chance that s is prime (and therefore, co-prime to r) is at least $1/2 \log(r) > 1/2 \log(N)$. Thus, repeating the algorithm $2 \log(N)$ times we will, with high probability, observe a phase s/r such that s and r are co-prime, and therefore the continued fraction algorithm produces r , as desired.

A second method is to note that if $r' \neq r$, then r' is guaranteed to be a factor of r , unless $s = 0$, which possibility occurs with probability $1/r \leq 1/2$, and which can be discounted further by a few repetitions. Suppose we replace a by $a' \equiv a^{r'} \pmod{N}$. Then the order of a' is r/r' . We can now repeat the algorithm, and try to compute the order of a' , which, if we succeed, allows us to compute the order of a since $r = r' \times r/r'$. If we fail, then we obtain r'' which is a factor of r/r' , and we now try to compute the order of a' , which, if we succeed, allows us to compute the order of a , since $r = r' \times r/r'$. If we fail, then we obtain r'' which is a factor of r/r' , and we now try to compute the order of $a'' = (a')^{r''} \pmod{N}$. We iterate this procedure until we determine the order of a . At most $\log(r) = \mathcal{O}(n)$ iterations are required, since each repetition reduces the order of the current candidate a'' by a factor of at least two.

The third method is better than the first two methods, in that it requires only a constant number of trials, rather than $\mathcal{O}(n)$ repetitions. The idea is to repeat the phase estimation -continued fractions procedure twice, obtain $r'_1 s'_1$ the first time, and r'_2, s'_2 takes the second time. Provided s'_1 and s'_2 have no common factors, r may be extracted by taking the least common multiple of r_1 and r_2 . The probability that s'_1 and s'_2 have not common factors satisfies

$$1 = \sum_q p(a|s'_1)p(1|s'_2) \geq 1 - \sum_1 \frac{1}{q^2}$$

The right hand side can be upper bounded in a number of ways, a simple technique is provided, which gives

$$1 - \sum_q p(q|s'_1)p(q|s'_2) \geq \frac{1}{4}$$

and thus the probability of obtaining the correct r is at least $1/4$.

Example 15.5.1. Show that the least common multiple of positive integers x and y is $xy/\gcd(x, y)$, and thus may be computed in $\mathcal{O}(n^2)$ operations if x and y are n bit numbers.

Example 15.5.2. For all $x \geq 2$ prove that $\int_x^{x+1} 1/y^2 dy \geq 2/3x^2$. Show that

$$\sum_q \frac{1}{q^2} \leq \frac{3}{2} \int_2^\infty \frac{1}{y^2} dy = \frac{3}{4}$$

So, by again setting $m = 2n$, applying phase estimation, and then applying the continued fraction algorithm to the result, we can find x and y such that

$$\frac{x}{y} = \frac{k}{r}$$

In other words, we can find k/r in lowest terms. this may fail to find r if $\gcd(k, r) > 1$, but we know that at least y divides r . Repeating several times (each time for a different k) and taking the least common multiple of the resulting y values gives r with high probability.

So, in the end, the algorithm is as follows. First choose $m = 2n$ and run the phase estimation procedure as shown in figure 15.8. (It can be run sequentially several times to improve accuracy.) Plug the measurement outcome $j/2^m$ along with N into the continued fraction algorithm, which gives a fraction x/y . Repeat the entire process several times, taking the least common multiple of the y values. The result will be r with high probability. the entire process is known as Shor's algorithm for order finding.

In performing the phase estimation, if we use $t = 2n + 1 + \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ ancilla qubits (in the first register), and prepare the second register in the state $|1\rangle$ which is trivial to construct - it follows that for each s in the range 0 to $r - 1$, we will obtain an estimate of the phase $\phi \approx s/r$ accurate to $2n + 1$ bits, with probability at least $(1 - \epsilon)/r$, where $n = \lfloor \log N \rfloor + 1$

15.6 Complexity Analysis

What resource requirement does the algorithm consume? The Hadamard transform requires $\mathcal{O}(n)$ gates, and the inverse Fourier transform requires $\mathcal{O}(n^2)$ gates. The major cost in the quantum circuit proper actually comes from the modular exponentiation, which uses $\mathcal{O}(n^3)$ gates, for a total of $\mathcal{O}(n^3)$ gates in the quantum circuit proper. The continued fractions algorithm adds $\mathcal{O}(n^3)$ more gates for a total of $\mathcal{O}(n^3)$, for a total of $\mathcal{O}(n^3)$ gates to obtain r' . Using the third method for obtaining r from r' we need only to repeat this procedure a constant number of time to obtain the order, r , for a total cost of $\mathcal{O}(n^3)$.

15.7 Summary

The following is the algorithm summarized.

Algorithm 3 Quantum order-finding Algorithm

- 1: **Input:** A black box $U_{x,N}$ which performs the transformation $|j\rangle|k\rangle \rightarrow |j\rangle|x^jk \bmod N\rangle$, for x co-prime to the n -bit number N , $t = 2n+1+\lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ qubits initialized to $|0\rangle$, and n qubits initialized to state $|1\rangle$.
 - 2: **Outputs:** The least integer $r > 0$ such that $x^r = 1 \pmod{N}$.
 - 3: **Runtime:** $\mathcal{O}(n^3)$ operations. Succeeds with probability $\mathcal{O}(1)$.
 - 4: **Procedure:**
 - 5: **Initial State:** $|0\rangle|1\rangle$
 - 6: **Create Superposition:** $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|1\rangle$
 - 7: **Apply $U_{x,N}$:** $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle \approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j / r} |j\rangle|u_s\rangle$
 - 8: **Apply inverse Fourier transform to first register** $\rightarrow \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\tilde{s}/r\rangle|u_s\rangle$
 - 9: **Measure first register** $|\tilde{s}/r\rangle$
 - 10: **Apply continued fractions algorithm** $\rightarrow r$
-

Chapter 16

Shor's Algorithm

16.1 Introduction

The most spectacular discovery in quantum computing to date is that quantum computers can effectively perform some tasks which are not feasible on a classical computer. For example, finding the prime factorization of an n -bit integer is thought to require $\exp(\Theta(n^{1/3} \log^{2/3} n))$ operations using the best classical algorithm known at the time of writing, the so-called number field sieve. This is exponential in the size of the number being factored, so factoring is generally considered to be an intractable problem on a classical computer: it quickly becomes impossible to factor even modest numbers. In contrast, a quantum algorithm can accomplish the same task using $O(n^2 \log n \log \log n)$ operations. That is, quantum computer can factor a number exponentially faster than the best known classical algorithms. This result is important in its own right, but perhaps the most exciting aspect is the question it raises: what other problems can be done efficiently on a quantum computer which are infeasible on a classical computer.

We can now factor integers efficiently using quantum computers and this is the problem solved by Shor's algorithm. Really, it turns out that "Shor's algorithm" is more of a technique than a specific algorithm, and it can be used to solve an interesting collection of problems. (Order finding is one of the simplest to describe.)

Let us now see that the integer factoring problem can be efficiently solved given an algorithm for order-finding. A fast algorithm for order-finding can easily be turned into a fast algorithm for factoring. In other words, factoring reduces to order finding. To be specific, it is a randomized polynomial-time Turing reduction, the types of reductions you might have seen. The idea, however, is essentially the same - if you have an efficient algorithm for order finding, then the reduction gives you an

efficient algorithm for factoring.

16.2 Problem Definition

Definition 16.2.1. Input: Given a number $N \geq 2$ which is a composite number.

Output: We are required to find the prime factorization of the number $N = p_1^{k_1} \dots p_m^{k_m}$.

Let us note a few facts, which we will not prove or discuss in any detail. first, if N is a prime number or a prime power (i.e. p^k for some prime p and integer $k \geq 1$), then there are efficient classical algorithms for solving the integer factorization problem in these case. So, we can imagine that we fist run such an algorithm on the input N ; if it succeeds then we are done, otherwise we continue on under the assumption that N has at least two distinct prime factors (i.e. $m \geq 2$).

Next, it is enough to have an algorithm that takes a composite N as input and just finds two integers $u, v \geq 2$ such that $N = uv$. If we have such an algorithm, we can run it recursively (interleaved with the classical algorithm for prime powers) to find a complete prime factorization of N .

Finally, if our goal is to find integers $u, v \geq 2$ such that $N = uv$ for N an even composite number, then we really don't need to work very hard; let $u = 2$ and $v = N/2$.

Now, consider the process described as follows. The basic idea is as follows. Suppose that the random choice of $a \in \mathbb{Z}_N^*$ (which is very likely), and that the order of a is even. Then

$$a^r \equiv 1 \pmod{N}$$

so

$$N | a^r - 1$$

because $a^r - 1 = (a^{r/2} + 1)(a^{r/2} - 1)$ we have

$$N | (a^{r/2} + 1)(a^{r/2} - 1)$$

It cannot happen that $N | (a^{r/2} - 1)$, because this would mean that r was not the order of a after all. If we are lucky and it is not the case than $N | (a^{r/2} + 1)$, then we know the algorithm will work. This is because the factors of N are then necessarily split better $(a^{r/2} + 1)$ and $(a^{r/2} - 1)$, so computing $\gcd(a^{r/2} - 1, N)$ will reveal a nontrivial factor of N .

Each iteration of the loop therefore fails to give an answer if either r is odd or r is even but N divides $a^{r/2} + 1$. The probability that neither of these events occurs is at

least $1/2$. (in fact, it is at least $1 - 2^{m-1}$ for m the number of distinct primes dividing N). This is why the assumption that N is not a prime power was important. Also, the analysis of this fact requires that N is odd.

16.3 Reduction of factoring to order-finding

The reduction of factoring to order-finding proceeds in two basic steps. The first step is to show that we can compute a factor of N if we can find a non-trivial solution $x \neq \pm 1 \pmod{N}$ to the equation $x^2 = 1 \pmod{N}$. The second step is to show that a randomly chosen y co-prime to N is quite likely to have an order r which is even, and such that $y^{r/2} \neq \pm 1 \pmod{N}$, and thus $x \equiv y^{r/2} \pmod{N}$ is a non-trivial solution to $x^2 = 1 \pmod{N}$. These two steps are embodied in the following theorems.

Theorem 16.3.1. *suppose N is an L bit composite number, and x is a non-trivial solution to the equation $x^2 = 1 \pmod{N}$ in the range $1 \leq x \leq N$, that is, neither $x = 1 \pmod{N}$ nor $x = N - 1 = -1 \pmod{N}$. Then at least one of $\gcd(x - 1, N)$ and $\gcd(x + 1, N)$ is a non-trivial factor of N that can be computed using $\mathcal{O}(L^3)$ operations.*

Theorem 16.3.2. *Suppose $N = p_1^{\alpha_1} \dots p_m^{\alpha_m}$ is the prime factorization of an odd composite positive integer. Let x be an integer chosen uniformly at random, subject to the requirements that $1 \leq x \leq N - 1$ and x is co-prime to N . Let r be the order of x modulo N . Then*

$$p(r \text{ is even and } x^{r/2} \neq -1 \pmod{N}) \geq 1 - \frac{1}{2^m}$$

Both these theorems can be combined to give an algorithm which, with high probability, returns a non-trivial factor of any composite N . All the steps in the algorithm can be performed efficiently on a classical computer except (so far as is known today) an order-finding ‘subroutine’ which is used by the algorithm. By repeating the procedure we may find a complete prime factorization of N . The algorithm is summarized as follows:

Steps 1 and 2 of the algorithm either return a factor, or else ensure that N is an odd integer with more than one prime factor. These steps may be performed using $\mathcal{O}(1)$ and $\mathcal{O}(L^3)$ operations, respectively. Step 3 either returns a factor, or produces a randomly chosen element x of $\{0, 1, 2, \dots, N - 1\}$. Step 4 calls the order-finding subroutine, computing the order r of modulo N . Step 5 completes the algorithm, since theorem guarantees that with probability at least one-half r will be even and

Algorithm 4 Shor's Algorithm

- 1: **Input:** A composite number N
 - 2: **Output:** A non-trivial factor of N
 - 3: **Runtime:** $\mathcal{O}((\log N)^3)$ operations. Succeeds with probability $\mathcal{O}(1)$.
 - 4:
 - If N is even, return the factor 2.
 - Determine whether $N = a^b$ for integers $a \geq 1$ and $b \geq 2$, and if so return the factor a (uses the classical algorithm).
 - Randomly choose x in the range 1 to $N - 1$. If $\gcd(x, N) > 1$ then return the factor $\gcd(x, N)$.
 - use the order-finding subroutine to find the order r of x modulo N .
 - If r is even and $x^{r/2} \neq -1 \pmod{N}$ then compute $\gcd(x^{r/2} - 1, N)$ and $\gcd(x^{r/2} + 1, N)$, and test to see if one of these is a non-trivial factor, returning that factor if so. Otherwise, the algorithm fails.
-

$x^{r/2} \neq -1 \pmod{N}$, and then theorem guarantees that either $\gcd(x^{r/2} - 1, N)$ or $\gcd(x^{r/2} + 1, N)$ is a non-trivial factor of N .

Example 16.3.1. Suppose N is L bits long. The aim of this exercise to ding an efficient classical algorithm to determine whether $N = a^b$ for some integers $a \geq 1$ and $b \geq 2$. This may be done as follows:

1. Show that b , if it exists, satisfies $b \leq L$.
2. Show that it takes at most $\mathcal{O}(L^2)$ to compute $\log_2 N, x = y/b$ for $b \leq L$, and the two integers u_1 and u_2 nearest to 2^x .
3. Show that it takes at most $\mathcal{O}(L^2)$ operations to compute u_1^b and u_2^b (use repeated squaring) and check to see if either is equal to N .
4. Combine the previous results to given an $\mathcal{O}(L^3)$ operation algorithm to determine whether $N = a^b$ for integers a and b .

Example 16.3.2. (Factoring 91) Suppose we wish to factor 91. Confirm that steps 1 and 2 are passed. For step 3, suppose we choose $x = 4$, which is co-prime to 91. Compute the order r of x with respect to N , and show that $x^{r/2} \pmod{91} = 64 \neq -1 \pmod{91}$, so the algorithm succeeds, giving $\gcd(64 - 1, 91) = 7$.

It is unlikely that this is the most efficient method you've seen for factoring 91. Indeed, if all computations had to be carried out on a classical computer, this reduction would not result in an efficient factoring algorithm, as no efficient method is known for solving the order-finding problem on a classical computer.

Example 16.3.3. Show that $N = 15$ is the smallest number for which the order-finding subroutine is required, that is, it is the smallest composite number that is not even or a power of some smaller integer.

16.4 Example

The use of order-finding, phase estimation, and continued fraction expansions in the quantum factoring algorithm is illustrated by applying it to factor $N = 15$. First, we choose a random number which has no common factors with N ; suppose we choose $x = 7$. Next, we compute the order r of x with respect to N , using the quantum order-finding algorithm: begin with the state $|0\rangle|0\rangle$ and create the state

$$\frac{1}{\sqrt{2^t}} \sum_{k=0}^{2^t-1} |k\rangle|0\rangle = \frac{1}{\sqrt{2^t}} [|0\rangle + |1\rangle + |2\rangle + \dots + |2^t-1\rangle] |0\rangle$$

by applying $t = 11$ hadamard transform to the first register. Choosing this value of t ensures an error probability ϵ of at most $1/4$. next, compute $f(k) = x^k \bmod N$, leaving the result in the second register,

$$\begin{aligned} & \frac{1}{\sqrt{2^t}} \sum_{k=0}^{2^t-1} |k\rangle|x^k \bmod N\rangle \\ &= \frac{1}{\sqrt{2^t}} [|0\rangle|1\rangle + |1\rangle|7\rangle + |2\rangle|4\rangle + |3\rangle|13\rangle + |4\rangle|1\rangle + |5\rangle|7\rangle + |6\rangle|4\rangle + \dots] \end{aligned}$$

We now apply the inverse Fourier transform FT^\dagger to the first register and measure it. One way of analyzing the distribution of outcomes obtained is to calculate the reduced density matrix for the first register, and apply FT^\dagger to it, and calculate the measurement statistics. However, since no further operation is applied to the second register, we can instead apply the principle of implicit measurement and assume that the second register is measured, obtaining a random result from $|1\rangle, |7\rangle, |4\rangle$ or $|13\rangle$. Suppose we get $|4\rangle$ (any of the result works); this means that the state input to FT^\dagger would have been $\sqrt{\frac{4}{2^t}} [|2\rangle + |6\rangle + |10\rangle + |14\rangle + \dots]$. After applying FT^\dagger we obtain

some state $\sum_l \alpha_l |l\rangle$, with the probability distribution 16.1 shown for $2^t = 2048$. The final measurement therefore gives either 0, 512, 1024 or 1536, each with probability almost exactly 1/4. Suppose we obtain $l = 1536$ from the measurement; computing the continued fraction expansion thus gives $1536/2048 = 1/(1 + (1/3))$, so that 3/4 occurs as a convergent in the expansion, giving $r = 4$ as the order of $x = 7$. By chance, r is even, and moreover, $x^{r/2} \bmod N = 7^2 \bmod 15 = 4 \neq -1 \bmod 15$, so the algorithm works: computing the greatest common divisor $\gcd(x^2 - 1, 15) = 3$ and $\gcd(x^2 + 1, 15) = 5$ tells us that $15 = 3 \times 5$.

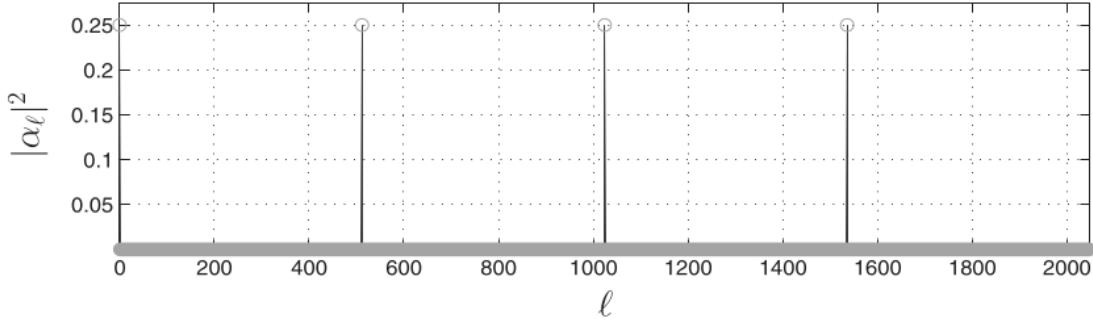


Figure 16.1: Plot of magnitude vs ℓ

16.5 Summary

The algorithm samples uniformly from \mathbb{Z}_m^* by sampling an a uniformly from the simple set $\{1, 2, \dots, m - 1\}$ and then checking whether or not $\gcd(a, m) = 1$. If $\gcd(a, m) = 1$ then $a \in \mathbb{Z}_m^*$ and the algorithm proceeds. If $\gcd(a, m) > 1$ then, although the algorithm could randomly sample another element from $\{1, 2, \dots, m - 1\}$, that's not necessary because, in that event, $\gcd(a, m)$ is a proper divisor of m .

In the event that $a \in \mathbb{Z}_m^*$, the probability that a is lucky is at least $\frac{1}{2}$. If a is lucky then the algorithm computes $\gcd(a^{r/2} + 1, m)$ to obtain a factor of m . This part can be computed by repeated squaring and Euclid's algorithm.

The implementation cost of this algorithm is $\mathcal{O}(n^2 \log n)$ gates. Although we only analyzed this algorithm for the case where m is the product of two distinct primes, it turns out that this factoring algorithm succeeds with probability at least $\frac{1}{2}$ for any number m that's not a prime power. For the prime power case (where $m = p^k$ for a prime p) there's a simple classical algorithm. That algorithm can be run as a preprocessing step to the algorithm.

Algorithm 5 Shor's Algorithm

- 1: **Input:** A composite number N
 - 2: **Output:** A non-trivial factor of N
 - 3: **Runtime:** $\mathcal{O}((\log N)^3)$ operations. Succeeds with probability $\mathcal{O}(1)$.
 - 4:
 - If N is even, return the factor 2.
 - Determine whether $N = a^b$ for integers $a \geq 1$ and $b \geq 2$, and if so return the factor a (uses the classical algorithm).
 - Randomly choose x in the range 1 to $N - 1$. If $\gcd(x, N) > 1$ then return the factor $\gcd(x, N)$.
 - use the order-finding subroutine to find the order r of x modulo N .
 - If r is even and $x^{r/2} \neq -1 \pmod{N}$ then compute $\gcd(x^{r/2} - 1, N)$ and $\gcd(x^{r/2} + 1, N)$, and test to see if one of these is a non-trivial factor, returning that factor if so. Otherwise, the algorithm fails.
-

Chapter 17

Period-finding

Consider the following problem.

17.1 Problem Definition

Suppose f is a periodic function producing a single bit as output such that $f(x+r) = f(x)$, for some unknown $0 < r < 2^L$, where $x, r \in \{0, 1, 2, \dots\}$. Given a quantum black box U which performs the unitary transform $U|x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle$ (where \oplus denotes addition modulo 2) how many black box queries and other operations are required to determine r ? Note that in practice U operates on a finite domain, whose size is determined by the desired accuracy of r . Here is a quantum algorithm which solve this problem using one query, and $\mathcal{O}(L^2)$ other operations.

The key to understanding this algorithm, which is based on phase estimation, and is nearly identical to the algorithm for quantum order finding, is step 3, in which we introduce the state

$$|\hat{f}(l)\rangle \equiv \frac{1}{\sqrt{r}} \sum_{x=0}^{r-1} e^{-2\pi\imath lx/r} |f(x)\rangle$$

the Fourier transform of $|f(x)\rangle$. The identity used in step 3 is based on

$$|f(x)\rangle = \frac{1}{\sqrt{r}} \sum_l -l = 0^{r-1} e^{2\pi\imath lx/r} |\hat{f}(l)\rangle$$

which is easy to verify by noting that $\sum_{l=0}^{r-1} e^{2\pi\imath lx/r} = r$ for x an integer multiple of r , and zero otherwise. The approximate equality in step 3 is required because 2^t may not be an integer multiple of r in general (it need not be: this is taken account of by the phase estimation bound). Applying the inverse Fourier transform to the first

Algorithm 6 Period-finding Algorithm

- 1: **Input:** (1) A black box which performs the operation $U |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$,
 (2) a state to store the function evaluation initialized to $|0\rangle$, and (3) $t = \mathcal{O}(L + \log(1/2\epsilon))$ qubits initialized to $|0\rangle$.
 - 2: **Output:** The least integer $r > 0$ such that $f(x + r) = f(x)$.
 - 3: **Runtime:** One use of U , and $\mathcal{O}(L^2)$ operations, succeeds with probability $\mathcal{O}(1)$.
 - 4: **Procedure:**
 - 5: **Initialize:** $|0\rangle |0\rangle$
 - 6: **Create superposition:** $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{x=0}^{2^t-1} |x\rangle |0\rangle$
 - 7: **Apply U :** $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{s=0}^{2^t-1} |x\rangle |f(x)\rangle \approx \frac{1}{\sqrt{r2^t}} \sum_{l=0}^{r-1} \sum_{x=0}^{2^t-1} e^{2\pi i l x / r} |x\rangle |\hat{f}(l)\rangle$
 - 8: **Apply inverse Fourier transform to first register** $\rightarrow \frac{1}{\sqrt{r}} \sum_{l=0}^{r-1} |l\rangle |\tilde{f}(l)\rangle$
 - 9: **Measure first register:** $\rightarrow l/\tilde{r}$
 - 10: **Apply continued fractions algorithm** $\rightarrow r$
-

register, in step 4, gives an estimate of the phase l/r , where l is chosen randomly, r can be efficiently obtained in the final step using a continued fraction expansion.

Chapter 18

Hidden Subgroup problem

One of the general application of the quantum Fourier transform are general problem known as the hidden subgroup problem. We will now describe an efficient quantum algorithm for solving it. This problem, which encompasses all known ‘exponentially fast’ application of the quantum Fourier transform, can be thought of as a generalization of the task of finding the unknown period of a periodic function, in a context where the structure of the domain and range of the function may be very intricate.

Chapter 19

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.

Suppose you are given a map containing many cities, and wish to determine the shortest route passing through all the cities on the map. A simple algorithm to find this route is to search all possible routes through the cities, keeping a running record of which route has the shortest length. On a classical computer, if there are N possible routes, it obviously takes $\mathcal{O}(N)$ operations to determine the shortest route using this method. Remarkably, there is a quantum search algorithm, sometimes known as Grover's algorithm, which enables this search method to sped up substantially, requiring only $\mathcal{O}(\sqrt{N})$ operations. Moreover, the quantum search algorithm is general in the sense that it can be applied far beyond the route-finding example just described to sped up many (though not all) classical algorithms that use search heuristics.

Three main applications of quantum search algorithm are: Quantum counting, speed-up of NP-complete problems, and search of unstructured databases. Here, we also show the optimality of grover's algorithm, i.e. no better search algorithm exists to do even better than a square root speedup. This speed limit applies to most unstructured problems.

19.1 The Problem Definition

Definition 19.1.1. The Search Problem For convenience let, $N = 2^n$, 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) U_f in the usual way in the computational basis as:

$$U_f |x\rangle |a\rangle = |x\rangle |a \oplus f(x)\rangle$$

for all $x \in \{0, 1\}^n$ and $a \in \{0, 1\}$, \oplus denotes addition modulo 2 and a is a single qubit which is flipped if $f(x) = 1$, and is unchanged otherwise. (We can build a quantum circuit for U_f given a Boolean circuit for f at a cost linear in size of boolean circuit). Here, we assume that we are supplied with a quantum oracle - a black box. 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. The goal of the algorithm is to find a solution to the search problem, using the smallest possible number of applications of the oracle.

Remark. Here, we assume that 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.

19.2 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. Then the probability that the k queries contain the solution x will be given by:

$$Pr = \frac{\binom{N-1}{k-1}}{\binom{N}{k}} = \frac{k}{N}$$

where $N = 2^n$, the number of ways in which our selection contains the solution (in this case we assume that there is a single value x for which $f(x) = 1$) is choosing the solution (1 way) times choosing $k - 1$ from $N - 1$ which is $\binom{N-1}{k-1}$ and the total number of ways of possible selections are $\binom{N}{k}$. Thus, the probability $\frac{\binom{N-1}{k-1}}{\binom{N}{k}} = \frac{k}{N}$. Now for the probability that our algorithm succeeds in finding x with probability at least $1 - \epsilon$ is,

$$\frac{k}{2^n} \geq 1 - \epsilon$$

On solving, we get, $k \geq (1 - \epsilon)2^n$ number of queries to the oracle. Thus, for constant error we need $\Omega(2^n) = \Omega(N)$ queries to solve the problem.

In the case, when there are multiple solutions (say a) and we are interested in designing a probabilistic classical algorithm that succeeds in finding all the solutions with probability at least $1 - \epsilon$ is given as follows:

$$Pr(\text{selecting all the solutions}) = \frac{\binom{N-a}{k-a}}{\binom{N}{k}} = \frac{k(k-1)\dots(k-a+1)}{n(n-1)\dots(n-a+1)}$$

The probability of selecting at least one of the correct solution can be similarly extended. **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 ($N = 2^n$).

19.3 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 4.7.

19.3.1 Idea of the Algorithm

Just a reminder, we can check whether x is a solution to our problem by preparing $|x\rangle|0\rangle$, applying the oracle, and checking to see if the oracle qubit has flipped to $|1\rangle$. 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 19.1 In the figure 19.1 it can be clearly seen that the $|A\rangle$ and $|B\rangle$

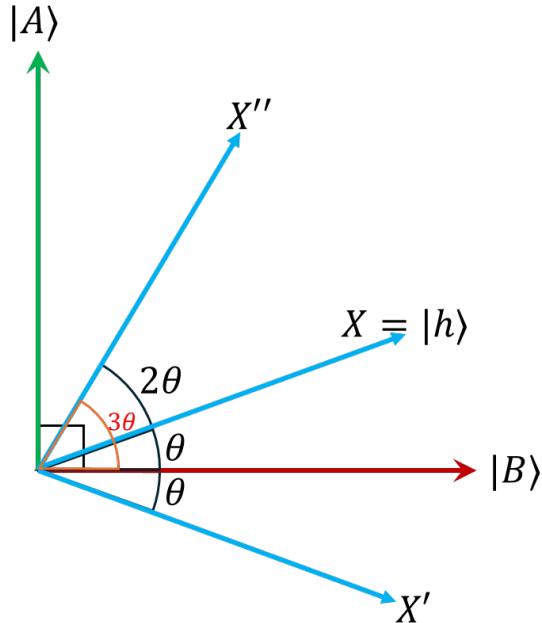


Figure 19.1: Geometric Visualization

are orthogonal. Say $X = |\hbar\rangle$ makes an angle θ 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 19.1, $|h\rangle$ 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 19.1 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 19.1 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 $|h\rangle$ which as shown in the figure 19.1 becomes $|X''\rangle$. Say this is denoted by the Unitary matrix $R_{|h\rangle}$. Thus, $|X''\rangle = R_{|h\rangle} |X'\rangle$

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 3θ 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.

19.3.2 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 19.1, 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**. The idea is we apply the oracle U_f with the oracle qubit in the state $(|0\rangle - |1\rangle)/\sqrt{2}$, just as was done in the Deutsch-Josza algorithm. If x is not a solution to the search problem, applying the oracle to the state $|x\rangle (|0\rangle - |1\rangle)/\sqrt{2}$ does not change the state. On the other hand, if x is a solution to the search problem, then $|0\rangle$ and $|1\rangle$ are interchanged by the action of the oracle, giving a final state $-|x\rangle (|0\rangle - |1\rangle)/\sqrt{2}$. The action of the oracle is thus,

$$|x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \xrightarrow{U_f} (-1)^{f(x)} \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

Notice that the state of the oracle qubit is not changed. It turns out that this remains $(|0\rangle - |1\rangle)/\sqrt{2}$ throughout the quantum search algorithm, and therefore be

omitted from further discussion of the algorithm, simplifying our description. With this convention, consider the following unitary transformation on n qubits:

$$Z_f |x\rangle = (-1)^{f(x)} |x\rangle$$

This is also called a *phase kickback*. We say that the oracle marks the solutions to the search problem, by shifting the phase of the solution. For an N item search problem with M solutions, it turns out that we need only apply this search oracle $\mathcal{O}(\sqrt{N/M})$ times in order to obtain a solution, on a quantum computer. 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). This discussion of the oracle without describing how it works in practice is rather abstract, and perhaps even puzzling. It seems as though the oracle already knows the answer to the search problem; what possible use could it be to have a quantum search algorithm based upon such oracle consultations?! The answer is that there is a distinction between knowing the solution to a search problem, and being able to recognize the solution; the crucial point is that it is possible to do the latter without necessarily being able to do the former.

Important Note

A simple example to illustrate this is the problem of factoring. Suppose we have been given a large number, m , and told that it is a product of two primes, p and q - the same sort of situation arises in trying to break the RSA public key cryptosystem. To determine p and q , the obvious method on a classical computer is to search all numbers from 2 through $m^{1/2}$ for the smaller of the two prime factors. That is, we successively do a trial division of m by each number in the range 2 to $m^{1/2}$, until we find the smaller prime factor. The other prime factor can be found by dividing m by the smaller prime. Obviously, this search-based method requires roughly $m^{1/2}$ trial divisions to find a factor on a classical computer.

The quantum search algorithm can be used to speed up this process. By definition, the action of the oracle upon input of the state $|x\rangle$ is to divide m by x , and check if the division is exact, flipping the oracle qubit if this is so. Applying the quantum search algorithm with this oracle yields the smaller of the two prime factors with high probability. But to make this algorithm work, we need to construct an efficient circuit implementing the oracle. How to do this? We begin by defining the function $f(x) \equiv 1$ if x divides m , and $f(x) = 0$ otherwise, $f(x)$ tells us whether the trial division is successful or not. Using the techniques of reversible computation, construct a reversible circuit which takes (x, q) - representing an input register initially set to x and a one bit output register initially set to q - to $(x, q \oplus f(x))$, by modifying the usual (irreversible) classical circuit for doing trial division. The resource cost of this reversible circuit is the same to within a factor two as the irreversible classical circuit used for trial division, and therefore we regard the two circuits as consuming essentially the same resources. Furthermore, the classical reversible circuit can be immediately translated into a quantum circuit that takes $|x\rangle|q\rangle$ to $|x\rangle|q \oplus f(x)\rangle$, as required of the oracle. The key point is that even without knowing the prime factors of m , we can explicitly construct an oracle which recognizes a solution to the search problem when it sees one. Using this oracle and the quantum search algorithm we can search the range 2 to $m^{1/2}$ using $\mathcal{O}(m^{1/4})$ oracle consultations. That is, we need only perform the trial division roughly $m^{1/4}$ times, instead of $m^{1/2}$ times, as with the classical algorithm!

The factoring example is conceptually interesting but not practical: there are classical algorithms for factoring which work much faster than searching through all possible divisors. However, it illustrates the general way in which the quantum search algorithm may be applied: classical algorithms which rely on search based techniques may be sped up using the quantum search algorithm. The quantum search algorithm offers a genuinely useful aid in speeding up the solution of **NP**-complete problems.

We now study the corresponding circuit as shown in the figure 19.2 which performs this transformation. Here, recall that the transformation B_f is the quantum

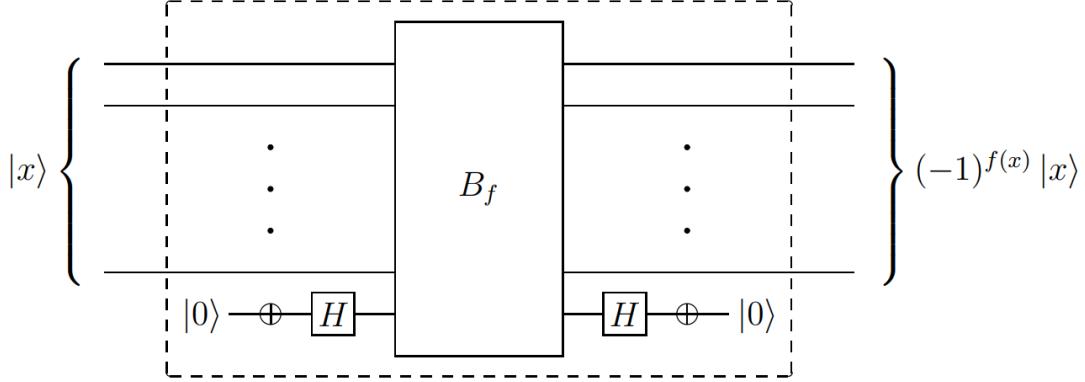


Figure 19.2: Reflection about B circuit

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 19.2.

$$(-1)^{f(x)} |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \xrightarrow{I^{\otimes n} \otimes H} (-1)^{f(x)} |x\rangle |1\rangle \xrightarrow{I^{\otimes n} \otimes X} (-1)^{f(x)} |x\rangle |0\rangle$$

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,

$$\begin{aligned} Z_f |X\rangle &= rZ_f |A\rangle + sZ_f |B\rangle \\ &= rZ_f \frac{1}{\sqrt{a}} \sum_{x \in A} |x\rangle + sZ_f \frac{1}{\sqrt{b}} \sum_{x \in B} |x\rangle \\ &= r \frac{1}{\sqrt{a}} \sum_{x \in A} Z_f |x\rangle + s \frac{1}{\sqrt{b}} \sum_{x \in B} Z_f |x\rangle \\ &= r \frac{1}{\sqrt{a}} \sum_{x \in A} -|x\rangle + s \frac{1}{\sqrt{b}} \sum_{x \in B} |x\rangle \\ &= -r|A\rangle + s|B\rangle \end{aligned}$$

As can be seen that the component of $|X\rangle$ along $|A\rangle$ is reversed. Thus, we achieve the reflection about $|B\rangle$.

Summarizing, recall that the algorithm begins with a superposition state. In order to create the superposition state $|h\rangle$, we start with $|0^n\rangle$ and apply the following hadamard transform $H^{\otimes n}$ on the state to achieve the superposition state

$$|h\rangle = H^{\otimes n} |0^n\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle = \frac{\sqrt{a}}{\sqrt{N}} |A\rangle + \frac{\sqrt{b}}{\sqrt{N}} |B\rangle$$

Now, since we already have the oracle B_f which performs the following operation $B_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$, then we create another oracle Z_F as follows. For the oracle operation to perform phase kickback, the ancilla bit is prepared in $|0\rangle$ and then acted upon by Pauli NOT X gate and then a Hadamard gate H and after the oracle operation B_f we perform uncomputation. Thus, achieving the required operation (ignoring the ancilla bit) as $Z_f |x\rangle = (-1)^{f(x)} |x\rangle$. Recall that we have already shown how to reflect any vector in the two dimensional space using the

reflection about $|B\rangle$ and thus we can start by reflection $|h\rangle$ in this two dimensional space about $|B\rangle$ and then later on repeat this step to reflect any vector about $|B\rangle$ in this two dimensional space repeatedly in order to achieve the final vector close to $|A\rangle$ (or exactly $|A\rangle$). The algorithm proper makes use of single n qubit register. The internal workings of the oracle, including the possibility of it needing extra work qubits, are not important to the description of the quantum search algorithm proper.

Reflection about $|h\rangle$

For the reflection about $|h\rangle$, we use the Diffusion operator D (assume $N = 2^n$), which works as follows. First, apply $H^{\otimes n}$, which maps $|h\rangle \rightarrow |00\dots 0\rangle$. 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 $|h\rangle$. 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,

$$D = H^{\otimes n} Z_0 H^{\otimes n} = H^{\otimes n} (I - 2|0^n\rangle\langle 0^n|) H^{\otimes n} = I - 2|h\rangle\langle h|$$

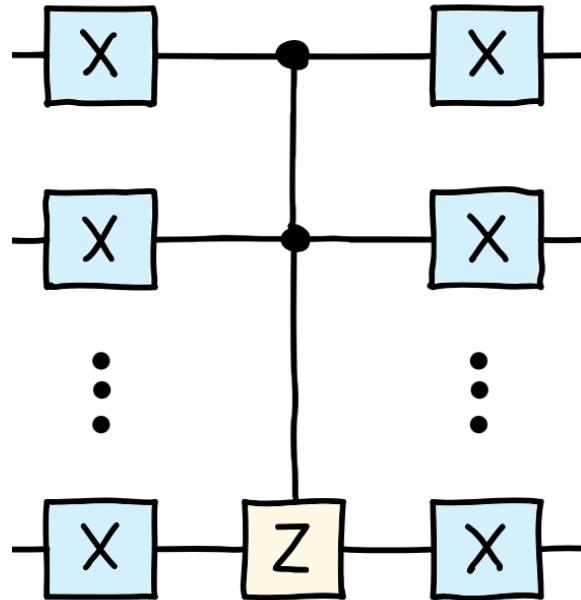
This is also the same as having a phase query gate Z_{OR} ($= -Z_0$) for the n-bit OR function:

$$OR(x) = \begin{cases} 0, & x = 0^n \\ 1, & x \neq 0^n \end{cases}$$

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. To perform the conditional phase shift operation $2|0^n\rangle\langle 0^n| - I$, we can implement the following circuit up to an unimportant global phase factor 19.3. Note that in this circuit the output on the input of $|0^n\rangle$ is $-|0^n\rangle$ and $|x\rangle$ where $x \neq 0^n$ is $|x\rangle$ which is opposite of what we wanted. Thus, the given circuit in figure 19.3 implements the $-(2|0^n\rangle\langle 0^n| - I) = I - 2|0^n\rangle\langle 0^n|$. But since this only affects the global phase factor and just negates the amplitudes, it does not change the probabilities. Also, we

Figure 19.3: General form of the operator $I - 2|0^n\rangle\langle 0^n|$

implement this operation k times, in case if k is even, the minus sign will vanish. In the case when k is odd and there will be a minus sign, still, the resulting state would just have negative amplitudes but since upon measurement the probabilities would not be affected and remain the same and also the result remains the same as global phase factor does not make any difference to the measurement. Thus, the complete conditional phase shift operator is as in figure 19.4: We now make the following propositions:

Proposition 19.3.1. *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

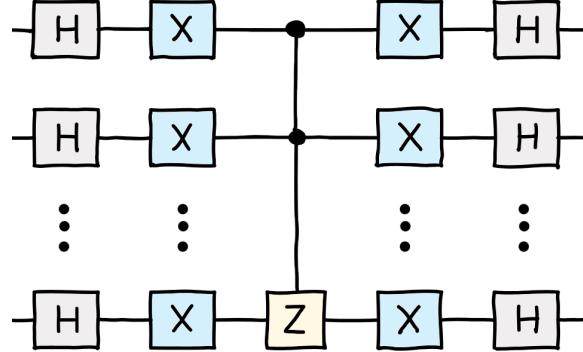


Figure 19.4: General conditional phase shift operator implementation $I - 2 |h\rangle \langle h|$

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. The Hadamard transform require $\mathcal{O}(\log N)$ operations each. The conditional phase shift may be implemented using $\mathcal{O}(n)$ gates. The cost of the oracle call depends upon the specific application.

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 α_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 α_i . □

Example 19.3.1. Show that the operation $(2|\psi\rangle\langle\psi| - I)$ applied to a general state $\sum_k \alpha_k |k\rangle$ produces

$$\sum_k (-\alpha_k + 2\langle\alpha\rangle) |k\rangle$$

where $\langle\alpha\rangle \equiv \sum_k \alpha_k / N$ is the mean value of the α_k . For this reason, $(2|\psi\rangle\langle\psi| - I)$ is sometimes referred to as the inversion about mean operation.

Recall that $|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$. Thus, $|\psi\rangle \langle \psi| = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} |x\rangle \langle y|$. Let us apply the operation as follows:

$$\begin{aligned}
(2|\psi\rangle \langle \psi| - I) \sum_k \alpha_k |k\rangle &= \left(2 \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} |x\rangle \langle y| \sum_k \alpha_k |k\rangle - \sum_k \alpha_k |k\rangle \right) \\
&= 2 \frac{1}{N} \sum_k \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \alpha_k |x\rangle \langle y| |k\rangle - \sum_k \alpha_k |k\rangle \\
&= 2 \frac{1}{N} \sum_k \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \alpha_k |x\rangle \delta_{y,k} - \sum_k \alpha_k |k\rangle \\
&= 2 \frac{1}{N} \sum_k \sum_{x=0}^{N-1} \alpha_k |x\rangle - \sum_k \alpha_k |k\rangle \\
&= 2 \sum_{x=0}^{N-1} \sum_k \frac{\alpha_k}{N} |x\rangle - \sum_k \alpha_k |k\rangle \\
&= 2 \sum_{x=0}^{N-1} \langle \alpha \rangle |x\rangle - \sum_k \alpha_k |k\rangle \\
&= 2 \langle \alpha \rangle \sum_{x=0}^{N-1} |x\rangle - \sum_{k=0}^{N-1} \alpha_k |k\rangle \\
&= 2 \langle \alpha \rangle \sum_{k=0}^{N-1} |k\rangle - \sum_{k=0}^{N-1} \alpha_k |k\rangle \\
&= \sum_{k=0}^{N-1} (2\langle \alpha \rangle - \alpha_k) |k\rangle \\
&= \sum_k (-\alpha_k + 2\langle \alpha \rangle) |k\rangle
\end{aligned}$$

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,

$$\begin{aligned} D|X'\rangle &= pD|h\rangle + qD|h^\perp\rangle \\ &= p(I - 2|h\rangle\langle h|)|h\rangle + q(I - 2|h\rangle\langle h|)|h^\perp\rangle \\ &= p(|h\rangle - 2|h\rangle\langle h|) + q(|h^\perp\rangle - 2|h\rangle\langle h|) \\ &= p|h\rangle - 2p|h\rangle + q|h^\perp\rangle \\ &= -p|h\rangle + q|h^\perp\rangle \end{aligned}$$

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 $|h\rangle$.

Example 19.3.2. Show that unitary operator corresponding to the phase shift in the Grover iteration is $2|0\rangle\langle 0| - I$.

We are interested in performing a conditional phase shift on the computer, with every computational basis except $|0\rangle$ receiving a phase shift of -1 . Let $|x\rangle$ be any n -qubit computational basis. Then, we have,

$$2|0\rangle\langle 0|x\rangle - |x\rangle$$

Say $|x\rangle = |0^n\rangle = |0\rangle$, then in that case we would have,

$$2|0\rangle\langle 0|0\rangle - |0\rangle = 2|0\rangle - |0\rangle = |0\rangle$$

For any other x i.e. $|x\rangle \neq |0^n\rangle$, we get,

$$2|0\rangle\langle 0|x\rangle - |x\rangle = -|x\rangle$$

since, $\langle 0|x\rangle = 0$ for $|x\rangle \neq |0^n\rangle = |0\rangle$. Hence, the unitary operator corresponding to the phase shift in the grover iteration is $2|0\rangle\langle 0| - I$.

Grover's Rotation Operator

The quantum search algorithm then consists of repeated application of a quantum subroutine, known as the Grover iteration or Grover operator, which we denote G . The Grover iteration may be broken into four steps:

1. Apply the oracle Z_f .
2. Apply the Hadamard transform $H^{\otimes n}$.
3. Perform a conditional phase shift on the computer, with every computational basis state except $|0\rangle$ receiving a phase shift of -1 .

$$|x\rangle \rightarrow -(-1)^{\delta_{x,0}} |x\rangle$$

4. Apply the Hadamard transform $H^{\otimes n}$.

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:

$$G = -DZ_f = -H^{\otimes n} Z_0 H^{\otimes n} Z_f$$

Example 19.3.3. Show that in the $|A\rangle, |B\rangle$ basis, we may write the Grover iteration as

$$G = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

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 for the basis of the matrix. 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 2θ in the space spanned by $\{|A\rangle, |B\rangle\}$ for

$$\theta = \sin^{-1} \sqrt{\frac{a}{N}}$$

Thus, this is the same as defining the Grover operator as follows:

$$G = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix}$$

where $\alpha = 2\theta = 2\sin^{-1} \sqrt{\frac{a}{N}}$. In other words, $\sin 2\theta = 2 \sin \theta \cos \theta$. Substituting $\sin \theta = \sqrt{\frac{a}{N}}$ and $\cos \theta = \sqrt{1 - \sin^2 \theta} = \sqrt{1 - \frac{a}{N}} = \sqrt{\frac{N-a}{N}}$. Thus, we get, $\sin 2\theta = 2 \frac{\sqrt{a(N-a)}}{N}$.

Now, we can write the state $|h\rangle$ as follows show in figure 19.5:

$$|h\rangle = \sqrt{\frac{b}{N}} |B\rangle + \sqrt{\frac{a}{N}} |A\rangle = \cos \theta |B\rangle + \sin \theta |A\rangle$$

Thus, the action of operator G on $|X\rangle$ for k times (i.e. after k iterations), the state

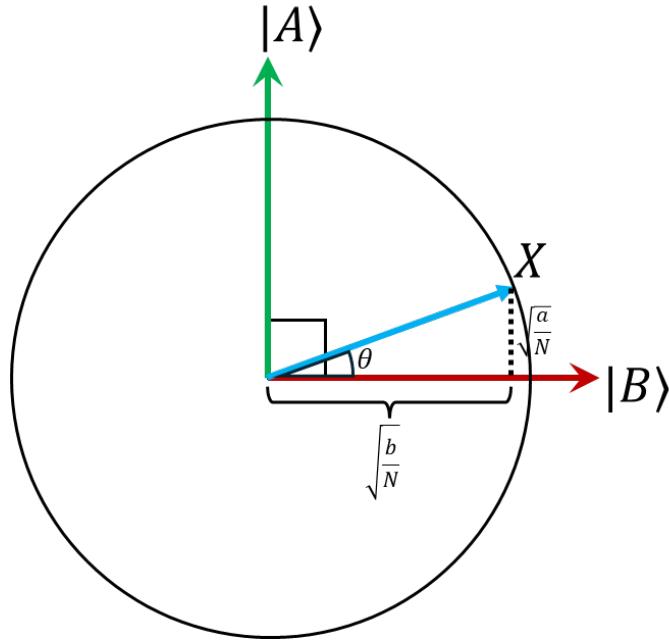


Figure 19.5: Action of G

$|X\rangle$ 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 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 19.6. 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.** Consider the two-dimensional plane. There's a transformation called a reflection about a line. The way it works is: every point on one side of the line is mapped to a mirror image point on the other side of the line,

Algorithm 7 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

as illustrated in figure 19.7. Now suppose that we add a second line of reflection, with angle θ between the lines. What happens if you compose the two reflections? That is, you apply reflection 1 and then reflection 2? Let the origin be where the two lines intersect and think of each point as a vector to that point. Now, start with any such vector. This vector makes some angle - call it θ_1 with the first line. After we perform the reflection, the reflected vector makes the same angle θ_1 on the other side of the line. The reflected vector makes the same angle with the second line. Call that angle θ_2 . Notice that $\theta_1 + \theta_2 = \theta$. Now, if we apply the second reflection then the twice reflected vector makes an angle θ_2 on the other side of the second line. So what happened to the original vector as a result of these two reflections? It has been rotated by angle 2θ . Notice that the amount of rotation depends only on θ , it does not depend on what θ_1 and θ_2 are.

Lemma 19.3.2. *For any two reflections with angle θ between them, their composition is a rotation by angle 2θ .*

A rigorous proof can be obtained by expressing each reflection operation as a 2×2 matrix, and then multiplying the two matrices. The result will be a rotation matrix by angle 2θ .

Schematically, the search algorithm operates as shown in figure 19.6. The algorithm proper makes use of a single n qubit register. The internal workings of the oracle, including the possibility of it needing extra work qubits, are not important to the description of the quantum search algorithm proper. Summarizing, G is a rotation in the two-dimensional space spanned by $|A\rangle$ and $|B\rangle$, rotating the space by θ radians per application of G . Repeated application of the Grover iteration rotates the state vector close to $|A\rangle$. When this occurs, an observation in the

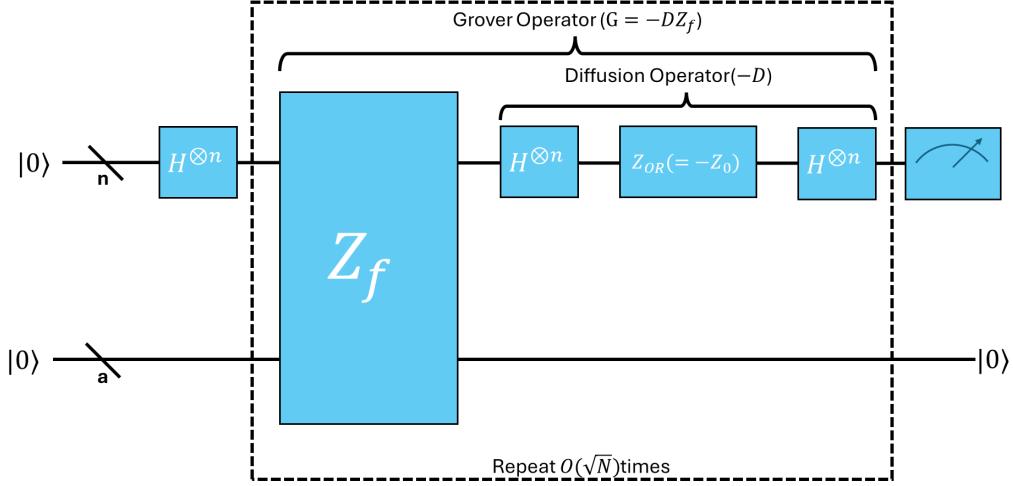


Figure 19.6: Grover's Circuit

computational basis produces with high probability one of the outcomes superposed in $|A\rangle$, that is, a solution to the search problem.

19.3.3 Example

For example, consider the function $f : \{0, 1\}^3 \rightarrow \{0, 1\}$ as shown in the table 19.1. Thus, the set $A = \{011, 100\}$ and $B = \{000, 001, 010, 101, 110, 111\}$. Let the Car-

Input	Output
000	0
001	0
010	0
011	1
100	1
101	0
110	0
111	0

Table 19.1: Example of Grover's Algorithm

dinality 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

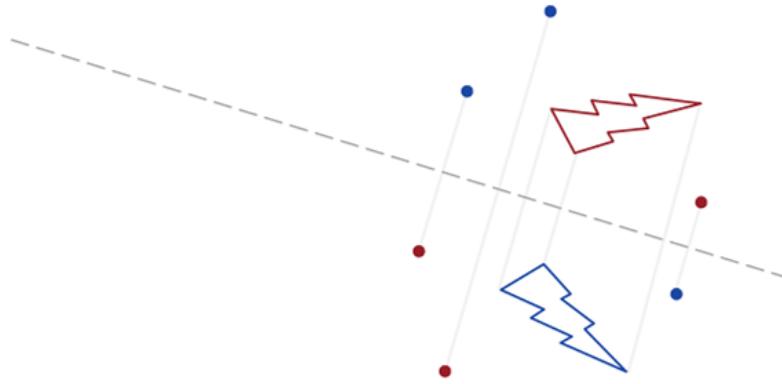


Figure 19.7: The reflection about the line of each red item is shown in blue

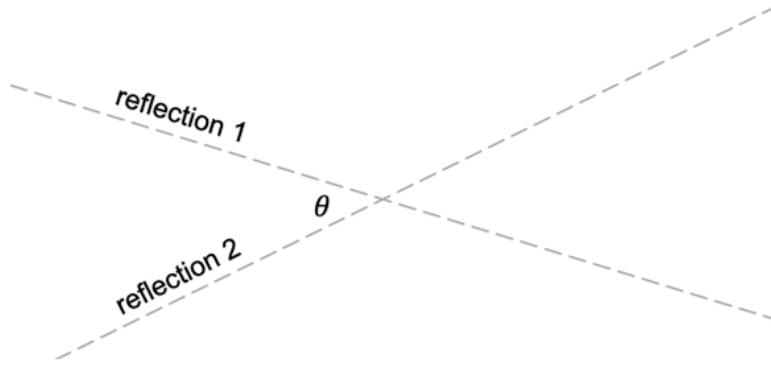


Figure 19.8: Two reflections with angle θ between them

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:

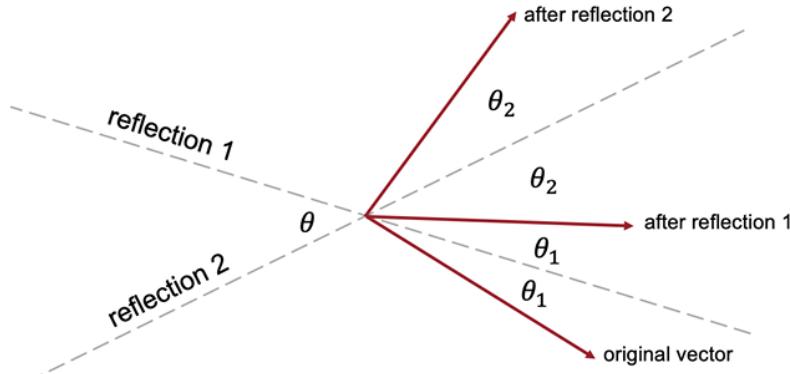


Figure 19.9: Effect of reflection 1 followed by a reflection 2 on a vector

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 $|h\rangle$:

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 θ to 3θ . 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.

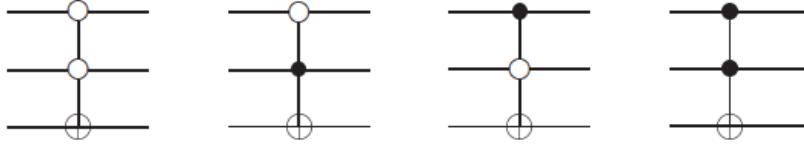
STEP4: MEASUREMENT:

Thus, supposed we pick $k = 1$ and then we perform the Grover's algorithm $k = 1$ times, we would get the state $|X''\rangle = |A\rangle$,

$$|X''\rangle = |A\rangle = \frac{1}{\sqrt{2}}(|011\rangle + |100\rangle)$$

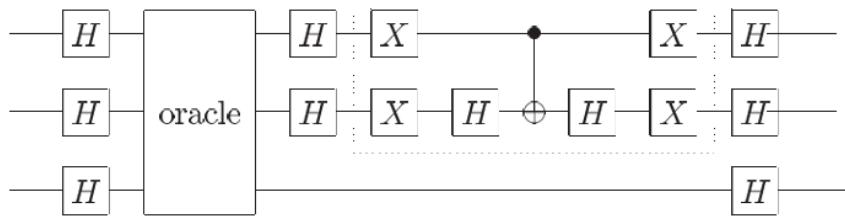
thus upon performing measurements we would get the required search strings $x = |100\rangle$ and $x = |011\rangle$ with the property that $f(x) = 1$ with equal probability. The Modified Grover's Algorithm is further explained in the following complexity analysis section (refer section 19.4).

Example 19.3.4. (Quantum search: a two-bit example) Here is an example illustrating how the quantum search algorithm works on a search space of size $N = 4$. The oracle, for which $f(x) = 0$ for all except $x = x_0$, in which case $f(x_0) = 1$, can be taken to be one of the four circuits corresponding to $x_0 = 0, 1, 2$ or 3 from left



to right, where the top two qubits carry the query x , and the bottom qubit carries the oracle's response. The quantum circuit which performs the initial Hadamard transform and a single Grover iteration G is

Initially, the top qubits are prepared in the state $|0\rangle$, and the bottom one as $|1\rangle$. The gates in the dotted box perform the conditional phase shift operation $|00\rangle \langle 00| - I$. How many times must we repeat G to obtain x_0 ? Using $M = 1$, we find that less than one iteration is required. It turns out that because $\theta = \pi/3$, only exactly one iteration is required to perfectly obtain, x_0 in this special case. In the geometric picture of figure 19.5, our initial state $|\psi\rangle = (|00\rangle + |01\rangle + |10\rangle + |11\rangle)/\sqrt{2}$ is 30 degree from $|B\rangle$, and a single rotation by 60 degree moves $|\psi\rangle$ to $|A\rangle$. You can confirm for yourself directly, using the oracle only once. In contrast, a classical computer, or a classical circuit - trying to differentiate between the four oracles would require on average 2.25 oracle queries!



19.4 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 θ which in turn depends on a which is the number of solutions. Thus, the number of iterations depends on number of solutions a but not on the identity of those 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.

19.4.1 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, for very small θ , $\sin^{-1} \theta \approx \theta \implies \sin^{-1} \frac{1}{\sqrt{N}} \approx \frac{1}{\sqrt{N}}$.

$$k \approx \frac{\pi}{4}\sqrt{N} - \frac{1}{2}$$

$$k \approx \frac{\pi}{4}\sqrt{N}$$

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 19.2

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

$$p(N, 1) \geq 1 - \frac{1}{N}$$

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 19.2: Unique Search ($a = 1$): Success Probability at various 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 .

19.4.2 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 19.3.

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 19.3: 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 θ , 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 .

$$\sin^{-1}(x) \geq x \quad \text{for every } x \in [0, 1]$$

$$\begin{aligned} \theta &= \sin^{-1} \left(\sqrt{\frac{a}{N}} \right) \geq \sqrt{\frac{a}{N}} \\ \implies k &\leq \frac{\pi}{4\theta} \leq \frac{\pi}{4} \sqrt{\frac{N}{a}} \\ k &= \mathcal{O} \left(\sqrt{\frac{N}{a}} \right) \end{aligned}$$

What happens when more than half the items are solutions to the search problem, that is $a \geq N/2$? From the expression $k = \lceil \frac{\pi}{4} \sqrt{\frac{N}{a}} \rceil$ and $\theta = \sin^{-1} \frac{\sqrt{a(N-a)}}{N}$ we see that the angle θ gets smaller as a varies from $N/2$ to N . As a result, the number of iterations needed by the search algorithm increases with a , for $a \geq N/2$. Intuitively, this is a silly property for a search algorithm to have: we expect that it should become easier to find a solution to the problem as the number of solutions increases.

Algorithm 8 Grover's Algorithm

-
- 1: **Input:**(1) A black box oracle O which performs the transformation $O|x\rangle|q\rangle = |x\rangle|q \oplus f(x)\rangle$, where $f(x) = 0 \forall 0 < x < 2^n$ except x_i for $1 \leq i \leq m$ for which $f(x_i) = 1$; (2) $n + 1$ qubits in the state $|0\rangle$.
- 2: **Output:** One of the x_i
- 3: **Runtime:** $\mathcal{O}(\sqrt{N/a})$ operations. Succeeds with probability $\mathcal{O}(1)$.
- 4: **Procedure:**
- 5: **Initial State** $|0\rangle^{\otimes n}|0\rangle$
- 6: **Apply $H^{\otimes n}$ to the first n qubits, and HX to the last qubit.** $\rightarrow \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$
- 7: **Apply the Grover iteration** $k = \lceil \frac{\pi}{4} \frac{\sqrt{2^n}a}{1} \rceil$ times.
- 8: $\rightarrow [(2|\psi\rangle\langle\psi| - I)O]^k \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$
- 9: $\approx \frac{1}{\sqrt{2^m}} \sum_{i=1}^m |x_i\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$
- 10: **Measure the first n qubits** $\rightarrow x_k$
-

there are at least two ways around this problem. If a is known in advance to be larger than $N/2$ then we can just randomly pick up an item in the search space, and then check that it is a solution using the oracle. This approach has a success probability at least one-half, and only requires one consultation with the oracle. It has the disadvantage that we may not know the number of solutions a in advance.

In the case where it isn't known whether $a \geq N/2$, another approach can be used. This approach is interesting in its own right, and has a useful application to simplify the analysis of the quantum algorithm for counting the number of solutions to the search problem. The idea is to double the number of elements in the search space by adding N extra items to the search space, none of which are solutions. As a consequence, less than half the items in the search space are solutions. This is effected by adding a single qubit $|q\rangle$ to the search index, doubling the number of items to be searched to $2N$. A new augmented oracle O' is constructed with marks an item only if it is a solution to the search problem and the extra bit is set to zero. The new scratch problem has only M solutions out of $2N$ entries, so running the search algorithm with the new oracle O' we see that at most $k = \frac{\pi}{4} \sqrt{2a/N}$ calls to O' are required, and it follows that $\mathcal{O}(\sqrt{N/a})$ calls to O are required to perform the search.

Example 19.4.1. Show that the augmented oracle O' may be constructed using one application of O , and elementary quantum gates, using the extra qubit $|q\rangle$.

The following circuit can be used: If $|q\rangle$ is set then on the second qubit we have

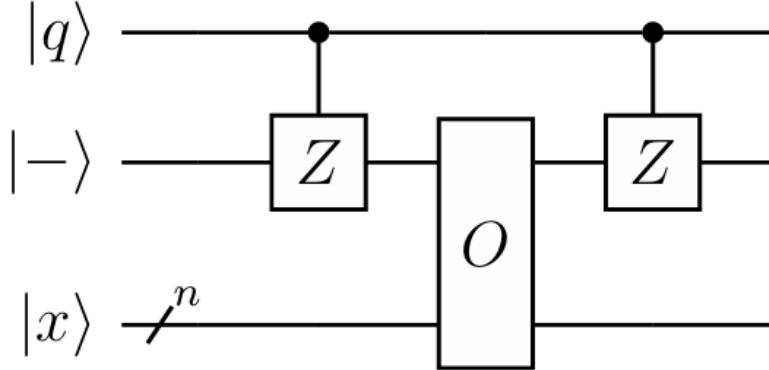


Figure 19.10: Circuit for oracle O'

$|- \rangle \rightarrow |+ \rangle$, hence the oracle performs $|x\rangle|+\rangle \rightarrow |x\rangle|+ \oplus f(x)\rangle = |x\rangle|+\rangle$ i.e. no solutions are marked. If the $|q\rangle$ is not set then the oracle functions as intended i.e. marking the correct solutions. This idea is also used in Quantum Counting.

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 9 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.

19.4.3 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.

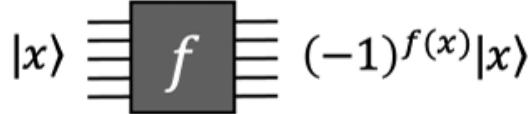
19.5 Optimality of Grover's algorithm

In this section we will prove that Grover's algorithm is optimal in terms of the number of black-box queries that it makes. We have shown that a quantum computer can search N items, consulting the search oracle only $\mathcal{O}(\sqrt{N})$ times. We now prove that no quantum algorithm can perform this task using fewer than $\Omega(\sqrt{N})$ accesses to the search oracle, and thus the algorithm we have demonstrated is optimal.

Theorem 19.5.1. (*optimality of Grover's algorithm*) *Any quantum algorithm for the search problem for functions of the form $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that succeeds with probability at least $\frac{3}{4}$ must make $\Omega(\sqrt{2^n})$ queries.*

Proof. We make the following two simplifying assumptions.

1. Assume that the function is always queried in the phase (as occurs with Grover's algorithm). To determine x we are allowed to apply an oracle Z_f which gives a phase shift of -1 to the solution and leaves all other states invariant, $Z_f = I - 2|x\rangle\langle x|$.



2. Assume that the algorithm uses only n qubits. That is, no ancilla qubits are used.

For simplicity, we prove the lower bound for the case where the search problem has a single solution, x . The purpose of these simplifying assumptions is to reduce clutter. It is very straightforward to adjust the proof to the general case, where the function is queried in the standard way, and where the quantum algorithm is allowed to use ancilla qubits. The more general proof will be just a more cluttered-up version of the proof about to be explained. Noting of importance is being swept under the rug in this simplified proof.

Figure 19.11: Form of a k query quantum circuit

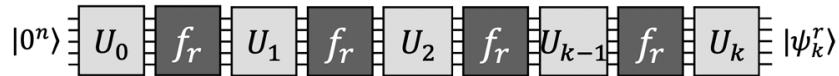
Under our assumptions, any quantum algorithm that makes k queries in a circuit is of the form 19.11. The initial state of the n qubits is $|0^n\rangle$. Then an arbitrary unitary operation U_0 is applied, which can create any pure state (say $|\psi\rangle$) as the input to the first query. Then the second f -query is performed. And so on, up to the k th f -query, followed by an arbitrary unitary operation U_k . The success probability is the probability that a measurement of the final state results in a satisfying input of f .

The quantum algorithm is the specification of the unitaries U_0, U_1, \dots, U_k . The input to the algorithm is the black box for f , which is inserted into the k query gates. The quantum output is the final state produced by the quantum circuit. The overall approach will be as follows. For every $r \in \{0, 1\}^n$, define f_r as

$$f_r(x) = \begin{cases} 1 & \text{if } x = r \\ 0 & \text{otherwise} \end{cases}$$

We'll show that, for any algorithm that makes asymptotically fewer than $\sqrt{2^n}$ queries, its success probability is very low for at least one f_r .

Assume that we have some k -query algorithm, specified by U_0, U_1, \dots, U_k . Suppose that we insert one of the function f_r into the query gate. Call the final state

Figure 19.12: Quantum circuit making k queries to the function f_r

$|\psi_k^r\rangle$ as shown in the figure 19.12. The superscript r is because this state depends on which r is selected. The subscript k is to emphasize that k queries are made.

$$|\psi_k^r\rangle = U_k f_r U_{k-1} f_r \dots U_1 f_r U_0 |0^n\rangle$$

That is, $|\psi_k^r\rangle$ is the state that results when the sequence of operations $U_0, f_r, U_1, f_r \dots, U_k$ is carried out. Now consider the exact same algorithm, run with identity operation in

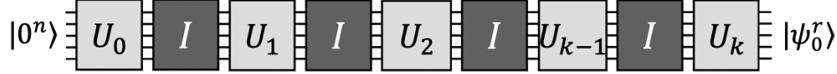


Figure 19.13: Quantum circuit with I substituted for the k queries

each query gate as shown in figure 19.13 instead of the oracle. Whenever the identity is substituted into a query gate, we'll call that a blank query. Let's call the final state produced by this $|\psi_0^r\rangle$. That is, $|\psi_0^r\rangle$ is the state that results when the sequence of operations U_0, U_1, \dots, U_k is carried out, without the oracle operations. Although the superscript r is redundant for this state (because the state is not a function of r), it's harmless and it will be convenient to keep this superscript r in our notation. The circuit if figure 19.12 corresponds to the function f_r . The circuit in figure 19.13 corresponds to the zero function (the function that's zero everywhere).

What we are going to show is that, for some $r \in \{0, 1\}^n$, the Euclidean distance between $|\psi_k^r\rangle$ and $|\psi_0^r\rangle$ is upper bounded as

$$\| |\psi_k^r\rangle - |\psi_0^r\rangle \| \leq \frac{2k}{\sqrt{2^n}}$$

This implies that, if k is asymptotically smaller than $\sqrt{2^n}$ then the bound asymptotically approaches zero. If $\| |\psi_k^r\rangle - |\psi_0^r\rangle \|$ approaches zero then the two states are not distinguishable in the limit. This implies that the algorithm cannot distinguish between f_r and the zero function with constant probability. Our goal will be to bound the quantity

$$D_k = \sum_r \| |\psi_k^r\rangle - |\psi_0^r\rangle \|^2$$

Intuitively, D_k is a measure of the deviation after k steps caused by the oracle, from the evolution that would otherwise have ensued. If this quantity is small, then all the states $|\psi_k^r\rangle$ are roughly the same, and it is not possible to correctly identify x with high probability. The strategy of the proof is to demonstrate two things: (a) a bound on D_k that shows it can grow no faster than $\mathcal{O}(k^2)$; and (b) a proof that D_k must be $\Omega(N)$ if it is possible to distinguish N alternatives. Combining these two results gives the desired lower bound. Now let's consider the quantum circuits in figures 19.12 and 19.13, along with some intermediate circuits as shown in figure 19.14. The circuit on top makes all k queries to f_r , and recall that we denote the final state that it produces as $|\psi_k^r\rangle$. The next circuit uses the identity in place of the first query and makes the remaining $k - 1$ queries to f_r . Call the final state of this $|\psi_{k-1}^r\rangle$. The next circuit uses the identity in place of the first two queries and makes

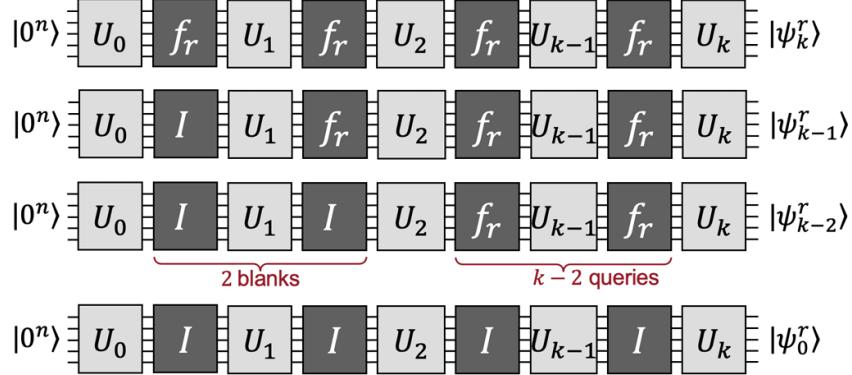


Figure 19.14: Quantum circuits with i blanks, followed by $k - i$ queries to f_r (for $i \in \{0, 1, \dots, r\}$)

the remaining $k - 2$ queries to f_r . Call the final state that this produces $|\psi_{k-2}^r\rangle$. And continue for $i = 3, \dots, k$ with circuit using the identity in place of the first i queries and then making the remaining $k - i$ queries to f_r . Call the final state $|\psi_{k-i}^r\rangle$.

First, we give an inductive proof that $D_k \leq 4k^2$. This is clearly true for $k = 0$, where $D_k = 0$. Note that

$$\begin{aligned} D_{k+1} &= \sum_r \|f_r |\psi_k^r\rangle - |\psi_0^r\rangle\|^2 \\ &= \sum_r \|f_r(|\psi_k^r\rangle - |\psi_0^r\rangle) + (f_r - I)|\psi_0^r\rangle\|^2 \end{aligned}$$

applying $\|b + c\|^2 \leq \|b\|^2 + \|c\|^2 + 2\|b\|\|c\|$ with $b \equiv f_r(|\psi_k^r\rangle - |\psi_0^r\rangle)$ and $c \equiv (f_r - I)|\psi_0^r\rangle = -2\langle x|\psi_k^r\rangle |x\rangle$, gives

$$D_{k+1} \leq \sum_r (\| |\psi_k^r\rangle - |\psi_0^r\rangle\|^2 + 4|\langle \psi_k^r | r \rangle|^2 + 4\| |\psi_k^r\rangle - |\psi_0^r\rangle \| |\langle r | \psi_0^r \rangle|)$$

Applying the Cauchy-Schwartz inequality to the second term on the right hand side, and noting that $\sum_r |\langle x | \psi_0^r \rangle|^2 = 1$ gives

$$\begin{aligned} D_{k+1} &\leq D_k + 4 \left(\sum_r \| |\psi_k^r\rangle - |\psi_0^r\rangle\|^2 \right)^{\frac{1}{2}} \left(\sum_{r'} |\langle \psi_0^r | r' \rangle|^2 \right)^{\frac{1}{2}} + 4 \\ &\leq D_k + 4\sqrt{D_k} + 4 \end{aligned}$$

By inductive hypothesis $D_k \leq 4k^2$ we obtain

$$D_{k+1} \leq 4k^2 + 8k + 4 = 4(k+1)^2$$

which completes the induction.

To complete the proof we need to show that the probability of success can only be high if D_k is $\Omega(N)$. We suppose $|\langle r|\psi_k^r \rangle|^2 \geq 1/2$ for all r , so that an observation yields a solution to the search problem with probability at least one-half. Replacing $|x\rangle$ by $e^{i\theta}|r\rangle$ does not change the probability of success, so without loss of generality we may assume that $\langle r|\psi_k^r \rangle = |\langle r|\psi_k^r \rangle|$, and therefore

$$\|\psi_k^r - \psi_0^r\|^2 = 2 - 2|\langle x|\psi_k^r \rangle| \leq 2 - \sqrt{2}$$

Example 19.5.1. Use the Cauchy-Schwarz inequality to show that for any normalized state vector $|\psi\rangle$ and set of N orthonormal basis vectors $|x\rangle$.

$$\sum_x \|\psi - x\|^2 \geq 2N - 2\sqrt{N}$$

Note that each query where the identity is substituted (blanks query) reveals no information about r . Recall that our goal is to show that the $\|\psi_k^r - \psi_0^r\|$ is small. Notice that we can express the difference between these states as the telescopic sum

$$|\psi_k^r\rangle - |\psi_0^r\rangle = (|\psi_k^r\rangle - |\psi_{k-1}^r\rangle) + (|\psi_{k-1}^r\rangle - |\psi_{k-2}^r\rangle) + \dots + (|\psi_1^r\rangle - |\psi_0^r\rangle)$$

which implies that

$$\|\psi_k^r - \psi_0^r\| = \|(|\psi_k^r\rangle - |\psi_{k-1}^r\rangle)\| + \|(|\psi_{k-1}^r\rangle - |\psi_{k-2}^r\rangle)\| + \dots + \|(|\psi_1^r\rangle - |\psi_0^r\rangle)\|$$

next, we'll upper bound each term $\|\psi_i^r - \psi_{i-1}^r\|$. To understand the Euclidean distance between $|\psi_i^r\rangle$ and $|\psi_{i-1}^r\rangle$, let's look at the circuits that produce them. Notice

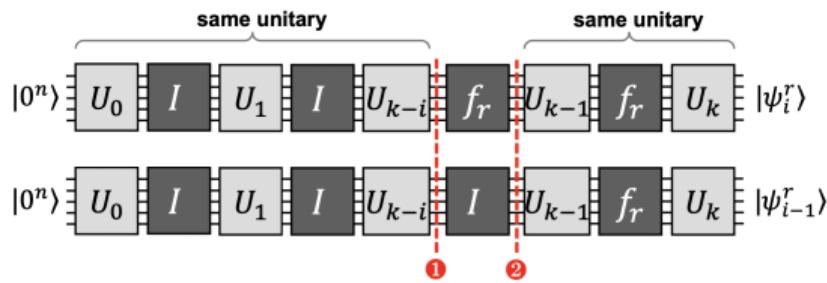


Figure 19.15: Circuit with $k - i$ blanks and i queries (top) and $k - i + 1$ blanks and $i - 1$ queries (bottom)

that both computations are identical for the first $k - i$ blank queries. We can write the state at stage 1 (for both the circuits) as

$$\sum_{x \in \{0,1\}^n} \alpha_{i,x} |x\rangle$$

where of course the state is a unit vector in Euclidean norm (a.k.a, the 2- norm)

$$\sum_{x \in \{0,1\}^n} |\alpha_{i,x}|^2 = 1$$

These states make sense for each $i \in \{1, \dots, k\}$, and they are independent of r .

It's interesting to consider what happens to the state at the next step .For the bottom computation, nothing happens to the state, since the identity is performed in the query. For the top computation, a query to f_r is performed. Since f_r takes value 1 only at the point r , the f_r -query in the phase negates the amplitude of $|r\rangle$ and has no effect on the other amplitudes of the state. Therefore, the state of the top computation at stage 2 is

$$\sum_{x \in \{0,1\}^n} (-1)^{[x=r]} \alpha_{i,x} |x\rangle$$

The difference between the state in the two equations is only in the amplitude of $|r\rangle$, which is negated. So at stage 2 the Euclidean distance between the states of the two circuits is $|\alpha_{r,i} - (-\alpha_{r,i})| = 2|\alpha_{r,i}|$.

Now let's look at what the rest of the two circuits in figure 19.15 do. The remaining steps for each circuit are the same unitary operation. Since unitary operations preserve Euclidean distances, this means that

$$\| |\psi_i^r\rangle - |\psi_{i-1}^r\rangle \| = 2|\alpha_{i,r}|$$

and this holds for all $i \in \{1, \dots, k\}$,

Now consider the average euclidean distance between $|\psi_k^r\rangle$ and $|\psi_0^r\rangle$, averaged over

all n -bit strings r . We can upper bound this as

$$\begin{aligned}
\frac{1}{2^n} \sum_{r \in \{0,1\}^n} \| |\psi_k^r\rangle - |\psi_0^r\rangle \| &\leq \frac{1}{2^n} \sum_{r \in \{0,1\}^n} \left(\sum_{i=1}^k \| |\psi_i^r\rangle - |\psi_{i-1}^r\rangle \| \right) \\
&= \frac{1}{2^n} \sum_{r \in \{0,1\}^n} \left(\sum_{i=1}^k 2|\alpha_{i,r}| \right) \\
&= \frac{1}{2^n} \sum_{i=1}^k 2 \left(\sum_{r \in \{0,1\}^n} |\alpha_{i,r}| \right) \\
&\leq \frac{1}{2^n} \sum_{i=1}^k 2 \left(\sqrt{2^n} \right) \\
&= \frac{2k}{\sqrt{2^n}}
\end{aligned}$$

□

using the Cauchy-Schwartz inequality, which implies that, for any vector whose 2- norm is 1, its maximum possible 1-norm is the square root of the dimension of the space, which in this case is $\sqrt{2^n}$.

Since an average of Euclidean distances is upper bounded by $2k/\sqrt{2^n}$, there must exist an r for which the Euclidean distance upper bounded by this amount.

Since the denominator is $\sqrt{2^n}$ and the numerator is $2k$, it must be the case that k is proportional to $\sqrt{2^n}$, in order for this Euclidean distance to be a constant. And the Euclidean must be lower bounded by a constant if the algorithm distinguishes between f_r and the zero function with probability $\frac{3}{4}$. This completes the proof that the number of queries k must be order $\sqrt{2^N}$.

This result, that the quantum search algorithm is essentially optimal, is both exciting and disappointing. It is exciting because it tells us that for this problem, at least, we have fully plumbed the depths of quantum mechanics; no further improvement is possible. the disappointment arises because we might have hoped to do much better than the square root speedup offered by the quantum search algorithm. the sort of dream result we might have hoped for a priori is that it would be possible to search an N item search space using $\mathcal{O}(\log N)$ oracle calls/ If such an algorithm existed, it would allow us to solve NP-complete problems efficiently on a quantum computer, since it could search all $2^{\omega(n)}$ possible witnesses using roughly $\omega(n)$ oracle calls, where the polynomial $\omega(n)$ is the length of a witness in bits. Unfortunately, such an algorithm is not possible. This is useful information for would-be

algorithm designers, since it indicates that a naive search-based method for attacking NP-complete problems is guaranteed to fail.

Venturing into the real of opinion, we note that many researchers believe that the essential reason for the difficulty of NP-complete problems is that their search space has essentially no structure, and that (up to polynomial factors) the best possible method for solving such a problem is to adopt a search method. If one takes this point of view, then sit is bad news for quantum computing, indicating that the class of problems efficiently soluble on a quantum computer, BQP does not contain the NP-complete problems. Of course, this is merely opinion, and it is still possible that the NP-complete problems contain some unknown structure that allows them to be efficiently solved on a quantum computer, or perhaps even on a classical computer. A nice example to illustrate this point is the problem of factoring, widely believed to be in the class of NPI problems of intermediate in difficulty between P and NP-complete problems. The key to the efficient quantum mechanical solution of the factoring problem was the exploitation of a structure ‘hidden’ within the problem - structure revealed by the reduction to order-finding. even with this amazing structure revealed, it has not been found possible to exploit the structure to develop an efficient classical algorithm for factoring, although, of course, quantum mechanically the structure can be harnessed to give an efficient factoring algorithm! perhaps a similar structure lurks in other problems b=suspected to be in NPI. such as the graph isomorphism problem, or perhaps even in the NP-complete problems themselves.

Example 19.5.2. (Optimality for multiple solutions) Suppose the search problem has M solutions. Show that $\mathcal{O}(\sqrt{N/M})$ oracle applications are required to find a solution.

19.6 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.

The great utility of this algorithm arises because we do not assume any particular structure to the search problems being performed. This is the great advantage of posing the problem in terms of a ‘black box’ oracle, and we adopt this point of view whenever convenient. In practical applications, of course, it is necessary to understand how the oracle is being implemented, and in each of the practical problems we concern ourselves with an explicit description of the oracle implementation is given.

19.7 Quantum Counting

How quickly can we determine the number of solutions, M , to an N item search problem, if M is not known in advance? Clearly, on a classical computer it takes $\theta(N)$ consultations with an oracle to determine M . On a quantum computer it is possible to estimate the number of solutions much more quickly than is possible on a classical computer by combining the Grover iteration with the phase estimation technique based upon the quantum Fourier transform. This has some important applications. First, if we can estimate the number of solutions quickly then it is possible to find a solution quickly, even if the number of solutions is unknown, by first counting the number of solutions, and then applying the quantum search algorithm to find a solution. Second, quantum counting allows us to decide whether or not a solution even exists, depending on whether the number of solutions is zero or non zero. This has applications, for example, to the solution of **NP**-complete problem, which may be phrased in terms of the existence of a solution to a search problem.

Example 19.7.1. Consider a classical algorithm for the counting problem which samples uniformly and independently k times from the search space, and let X_1, \dots, X_k be the results of the oracle calls, that is, $X_j = 1$ if the j th oracle call revealed a solution to the problem, and $X_j = 0$ if the j th oracle call did not reveal a solution to the problem. This algorithm returns the estimate $S = N \times \sum_j X_j/k$ for the number of solutions to the search problem. Show that the standard deviation is $\Delta S = \sqrt{a(N-a)/k}$. Prove that to obtain a probability at least $3/4$ of estimating M correctly to within an accuracy \sqrt{M} for all values of M must have $k = \Omega(\sqrt{N})$.

Example 19.7.2. Prove that any classical counting algorithm with a probability at least $3/4$ for estimating M correctly to within an accuracy $c\sqrt{M}$ for some constant c and for all values of M must make $\Omega(N)$ oracle calls.

Quantum counting is an application of the phase estimation procedure to estimate the eigenvalues of the grover iteration G , which in turn enables us to determine the number of solutions M to the search problem. Suppose $|a\rangle$ and $|b\rangle$ are the two

eigenvectors of Grover iteration in the space spanned by $|A\rangle$ and $|B\rangle$. Let θ be the angle of rotation determined by the Grover iteration. From equation,

$$G = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

The characteristic equation is as follows:

$$\lambda^2 - 2 \cos \theta \lambda + 1 = 0$$

The solutions to these equations are:

$$\lambda = \frac{2 \cos \theta \pm \sqrt{4 \cos^2 \theta - 1}}{2} = \cos \theta \pm \sqrt{-\sin^2 \theta} = e^{i\theta}, e^{-i\theta}$$

we see that the corresponding eigen values are $e^{i\theta}$ and $e^{i(2\pi-\theta)}$. For ease of analysis it is convenient to assume that the oracle has been augmented, expanding the size of the search space to $2N$, and ensuring that $\sin^2(\theta/2) = a/2N$.

The phase estimation circuit used for quantum counting is as shown in figure 19.16. The function of the circuit is to estimate θ to m bits of accuracy, with a probability of success at least $1-\epsilon$. The first register contains $t = m + \lceil \log(2 + 1/\epsilon) \rceil$ qubits, as per the phase estimation algorithm, and the second register contains $n+1$ qubits, enough to implement the Grover iteration on the augmented search space. The state of the second register is initialized to an equal superposition of all possible inputs $\sum_x |x\rangle$ by a Hadamard transform. As we saw this state is a superposition of the eigenstates $|a\rangle$ and $|b\rangle$, so the circuit gives us an estimate for θ or $2\pi - \theta$ accurate to within $|\Delta\theta| \leq 2^{-m}$, with probability at least $1 - \epsilon$ with the same level of accuracy, so effectively the phase estimation algorithm determines θ to an accuracy of 2^{-m} with probability $1 - \epsilon$. Using the equation $\sin^2(\theta/2) = a/2N$ and our estimate for θ we obtain an estimate of the number of solutions, a . How large an error, ΔM , is there in this estimate

$$\begin{aligned} \frac{|a|}{2N} &= \left| \sin^2 \left(\frac{\theta + \Delta\theta}{2} \right) - \sin^2 \left(\frac{\theta}{2} \right) \right| \\ &= \left(\sin \left(\frac{\theta + \Delta\theta}{2} \right) + \sin \left(\frac{\theta}{2} \right) \right) \left| \sin \left(\frac{\theta + \Delta\theta}{2} \right) - \sin \left(\frac{\theta}{2} \right) \right| \end{aligned}$$

Calculus implies that $|\sin((\theta + \Delta\theta)/2) - \sin(\theta/2)| \leq |\Delta\theta|/2$, and elementary trigonometry that $|\sin((\theta + \Delta\theta)/2)| \leq \sin(\theta/2) + |\Delta\theta|/2$, so

$$\frac{|\Delta a|}{2N} < \left(2 \sin \left(\frac{\theta}{2} \right) + \frac{|\Delta\theta|}{2} \right) \frac{|\Delta\theta|}{2}$$

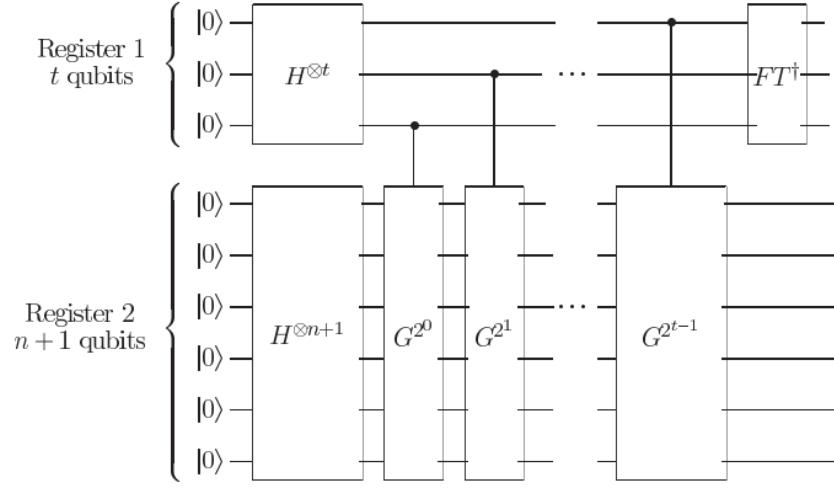


Figure 19.16: Circuit for performing approximate quantum counting on a quantum computer

Substituting $\sin^2(\theta/2) = a/2N$ and $|\Delta\theta| \leq 2^{-m}$ gives our final estimate for the error in our estimate of M ,

$$|\Delta a| < \left(\sqrt{2aN} + \frac{N}{2^{m+1}} \right) 2^{-m}$$

As an example, suppose we choose $m = \lceil n/2 \rceil + 1$, and $\epsilon = 1/6$. Then we have $t = \lceil n/2 \rceil + 3$, so the algorithm requires $\Theta(\sqrt{N})$ Grover iterations, and thus $\Theta(\sqrt{N})$ oracle calls. Our accuracy is $|\Delta M| < \sqrt{M/2} + 1/4 = \mathcal{O}(\sqrt{M})$. Comparing this with the fact that it would have required $\mathcal{O}(N)$ oracle calls to obtain a similar accuracy on a classical computer.

Indeed, the example just described serves double duty as an algorithm for determining whether a solution to the search problem exists at all, that is, whether $M = 0$ or $M \neq 0$. If $M = 0$ then we have $|\Delta M| < 1/4$, so the algorithm must produce the estimate zero with probability at least $5/6$. Conversely, if $M \neq 0$ then it is easy to verify that the estimate for M is not equal to 0 with probability at least $5/6$.

Another application of quantum counting is to find a solution to a search problem when the number M of solutions is unknown. The difficulty in applying the quantum search algorithm is that the number of time to repeat the Grover iteration, depends on knowing the number of solutions M . This problem can be alleviated by using the quantum counting algorithm to first estimate θ and M to high accuracy using phase estimation, and then to apply the quantum search algorithm, repeating the Grover

iteration a number of times determined, with the estimates for θ and M obtained by the phase estimation substituted to determine R . The angular error in this case is at most $\pi/4(1 + |\Delta\theta|/\theta)$, so choosing $m = \lceil n/2 \rceil + 1$ as before gives an angular error at most $\pi/4 \times 3/2 = 3\pi/8$, which corresponds to a success probability of at least $\cos^2(3\pi/8) = 1/2 - 1/2\sqrt{2} \approx 0.15$ for the search algorithm. If the probability of obtaining an estimate of θ this accurate is $5/6$, as in our earlier example, then the total probability of obtaining a solution to the search problem is $5/6 \times \cos^2(3\pi/8) \approx 0.12$, a probability which may be boosted close to 1 by a few repetitions of the combined counting-search procedure.

19.8 Amplitude Estimation

Let $|\psi_0\rangle$ be prepared by an oracle U_{ψ_0} i.e., $U_{\psi_0}|0^n\rangle = |\psi_0\rangle$. We have the knowledge that

$$|\psi_0\rangle = \sqrt{p_0}|\psi_{good}\rangle + \sqrt{1-p_0}|\psi_{bad}\rangle$$

and $p_0 \ll 1$. Here $|\psi_{bad}\rangle$ is an orthogonal state to the desired state $|\psi_{good}\rangle$, and $\sqrt{p_0} = \sin \frac{\theta}{2}$. The problem of amplitude estimation is to estimate p_0 to ϵ -precision. If p_0 is away from 0 or 1 and is to be estimated directly from the Monte Carlo methods, the number of samples needed is $N = O(\epsilon^{-2})$.

Let $G = R_{\psi_0}R_{good}$ be the grover operator. Then in the basis $B = \{|\psi_{good}\rangle, |\psi_{bad}\rangle\}$, the subspace $H = \text{span}(B)$ is an invariant subspace of G . The matrix representation is

$$[G]_B = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix}$$

Its two eigenstates are

$$|\psi_{\pm}\rangle = \frac{1}{\sqrt{2}}(|\psi_{good}\rangle \pm \iota |\psi_{bad}\rangle)$$

with the eigenvalues $e^{\pm\theta}$, respectively.

Therefore the problem of estimation θ can be solved with phase estimation with an imperfect initial state. Note that

$$|\langle\psi_0|\psi_+\rangle|^2 = \frac{1}{2} \left| \sin \frac{\theta}{2} + \iota \cos \frac{\theta}{2} \right|^2 = \frac{1}{2} = |\langle\psi_0|\psi_-\rangle|^2$$

Consider a QPE circuit with t ancilla qubits and query G for $T = 2^t$ times. Then each execution with the system register will be $|\psi_+\rangle$ or $|\psi_-\rangle$ states with probability 0.5. let

$$t = d + \lceil \log \delta^{-1} \rceil$$

be the number of ancilla qubits with $\epsilon' = 2^{-d}$. Then QPE obtains an estimate denoted by $\tilde{\theta}$, which approximates θ to precision ϵ' with success probability $1 - \delta$. Note that $p_0 = \sin^2 \frac{\theta}{2}$, and

$$\begin{aligned} & \sin^2 \frac{\tilde{\theta}}{2} - \sin^2 \frac{\theta}{2} \\ &= \sin^2 \frac{\tilde{\theta} - \theta}{2} \cos^2 \frac{\theta}{2} + \cos^2 \frac{\tilde{\theta} - \theta}{2} \sin^2 \frac{\theta}{2} + 2 \sin \frac{\theta}{2} \cos \frac{\theta}{2} \sin \frac{\tilde{\theta} - \theta}{2} \cos \frac{\tilde{\theta} - \theta}{2} - \sin^2 \frac{\theta}{2} \\ &= \sin \frac{\theta}{2} \cos \frac{\theta}{2} \sin(\tilde{\theta} - \theta) + \left(1 - 2 \sin^2 \frac{\theta}{2}\right) \sin^2 \frac{\tilde{\theta} - \theta}{2} \end{aligned}$$

Using the fact that $|\sin(\tilde{\theta} - \theta)| \leq |\tilde{\theta} - \theta| \leq \epsilon'$, we have

$$|\tilde{p} - p| \leq \sqrt{p_0(1-p_0)}\epsilon' + (1-2p_0)\frac{\epsilon'^2}{4}$$

Let ϵ' be sufficiently small. Now if $p_0(1-p_0) = \Omega(1)$, we can choose $\epsilon' = \mathcal{O}(\epsilon)$, and the total complexity of QPE is $\mathcal{O}(\epsilon^{-1})$.

If p_0 is small, then we should estimate p_0 to multiplicative accuracy ϵ instead. Use

$$|\tilde{p} - p| \approx \sqrt{p_0}\epsilon' < p_0\epsilon$$

we have $\epsilon' = \sqrt{p_0}\epsilon$. Therefore the runtime of QPE is $\mathcal{O}(p_0^{-\frac{1}{2}}\epsilon^{-1})$. If p_0 is to be estimated to precision ϵ' using Monte Carlo methods, the number of samples needed would be $\mathcal{N} = \mathcal{O}(p_0^{-1}\epsilon^{-2})$.

So, the amplitude estimation method achieves quadratic speedup in the total complexity, by the circuit depths is increased to $\mathcal{O}(\epsilon'^{-1}\delta^{-1})$.

Example 19.8.1. (Amplitude estimation to accelerate hadamard test) Consider the circuit for the Hadamard test to estimate $\text{Re} \langle \psi | U | \psi \rangle$. Let the initial state $|\psi\rangle$ be prepared by a unitary U_ψ , then the following combined circuit 19.17. maps $|0\rangle |0^n\rangle$ to

$$|\psi_0\rangle = \frac{1}{2} |0\rangle (|\psi\rangle + U|\psi\rangle) + \frac{1}{2} |1\rangle (|\psi\rangle - U|\psi\rangle) = \sqrt{p_0} |\psi_{good}\rangle + \sqrt{1-p_0} |\psi_{bad}\rangle$$

and the goal is to estimate p_0 . This also gives the implementation of the reflector R_{ψ_0} .

Note that R_{ψ_0} can be implemented by simply reflecting against the signal qubit, i.e.,

$$R_{\psi_0} = (I - 2|0\rangle\langle 0|) \otimes I_n = -Z \otimes I_n$$

Then we can run QPE to the Grover unitary $G = R_{\psi_0} R_{\psi_{good}}$ to estimate $p(0)$, and the circuit depth is $\mathcal{O}(\epsilon^{-1})$.

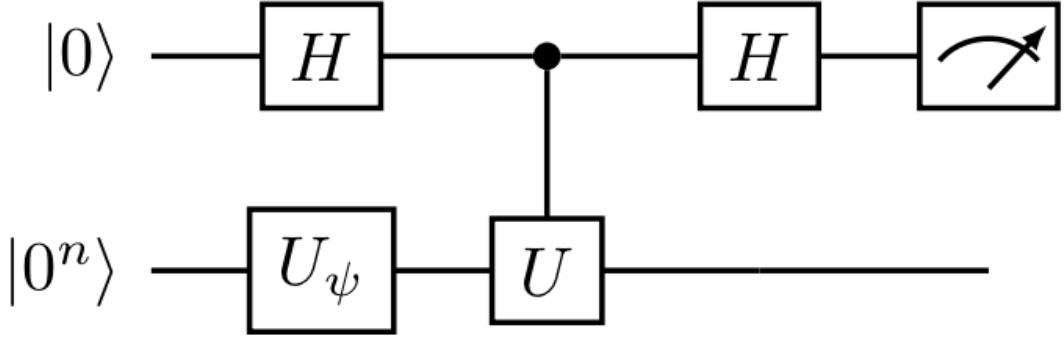


Figure 19.17: Accelerated Hadamard Test

19.9 Amplitude Amplification

Grover's algorithm is not restricted to the problem of unstructured search. One immediate application is called amplitude amplification (AA), which is used ubiquitously as a subroutine to achieve quadratic speedups.

Let $|\psi_0\rangle$ be prepared by an oracle U_{ψ_0} i.e. $U_{\psi_0} = |0^n\rangle = |\psi_0\rangle$. We have the knowledge that

$$|\psi_0\rangle = \sqrt{p_0} |\psi_{good}\rangle + \sqrt{1-p_0} |\psi_{bad}\rangle$$

and $p_0 \ll 1$. Here $|\psi_{bad}\rangle$ is an orthogonal state to the desired state $|\psi_{good}\rangle$. We cannot get access to $|\psi_{good}\rangle$ directly, but would like to obtain a state that has a large overlap with $|\psi_{good}\rangle$ i.e., amplify the amplitude of $|\psi_{good}\rangle$.

In the problem of unstructured search $|\psi_{good}\rangle = |x_0\rangle$, and $p_0 = 1/N$. Although we do not have access to the answer $|x_0\rangle$, we assume access to the reflection oracle R_{x_0} . Here we also assume access to the reflection oracle

$$R_{good} = 1 - 2 |\psi_{good}\rangle \langle \psi_{good}|$$

From U_{ψ_0} , we can construct the reflection with respect to the initial state

$$R_{\psi_0} = 2 |\psi_0\rangle \langle \psi_0| - I = U_{\psi_0} (2 |0^n\rangle \langle 0^n| - I) U_{\psi_0}^\dagger$$

via the n -qubit controlled- X gate. So following exactly the same procedure as the unstructured search problem, we can construct the Grover iterate

$$G = R_{\psi_0} R_{good}$$

Applying G^k to $|\psi_0\rangle$ for some $k = \mathcal{O}(1/\sqrt{p_0})$, we obtain a state that has $\Omega(1)$ overlap with $|\psi_{good}\rangle$.

Example 19.9.1. (Reflection with respect to signal qubits) One common scenario is that the implementation of U_{ψ_0} requires m ancilla qubits (also called signal qubits) i.e.,

$$U_{\psi_0} |0^m\rangle |0^n\rangle = \sqrt{p_0} |0^m\rangle |\psi_0\rangle + \sqrt{1-p_0} |\perp\rangle$$

where $|\perp\rangle$ is some orthogonal state satisfying

$$(\Pi \otimes I_n) |\perp\rangle = 0; \quad \Pi = |0^m\rangle \langle 0^m|$$

Therefore

$$|\psi_{good}\rangle = |0^m\rangle |\psi_0\rangle$$

This setting is special since the “good” state can be verified by measuring the ancilla qubits after applying U_{ψ_0} , and post-select the outcome 0^m . In particular, the expected number of measurements needed to obtain $|\psi_{good}\rangle$ is $1/p_0$.

In order to employ the AA procedure, we first note that the reflection operator can be simplified as

$$R_{good} = (1 - 2|0^m\rangle \langle 0^m|) \otimes I_n = (I - 2\Pi) \otimes I_n$$

This is because $|\psi_{good}\rangle$ can be entirely identified by measuring the ancilla qubits. Meanwhile

$$R_{\psi_0} = U_{\psi_0} (2|0^{m+n}\rangle \langle 0^{m+n}| - I) U_{\psi_0}^\dagger$$

Let $G = R_{\psi_0} R_{good}$, and applying G^k to $|\psi_0\rangle$ for some $k = \mathcal{O}(1/\sqrt{p_0})$ times, we obtain a state that has $\Omega(1)$ overall with $|\psi_{good}\rangle$. This achieves the desired quadratic speedup.

Example 19.9.2. (Amplitude Damping) Assuming access to an oracle, where p_0 is large, we can easily dampen the amplitude to any number $\alpha \leq \sqrt{p_0}$.

We introduce an additional signal qubit. Then,

$$(I \otimes U_{\psi_0}) |0\rangle |0^m\rangle |0^n\rangle = |0\rangle (\sqrt{p_0} |0^m\rangle |\psi_0\rangle + \sqrt{1-p_0} |\perp\rangle)$$

Define a single qubit rotation operation as

$$R_\theta |0\rangle = \cos \theta |0\rangle + \sin \theta |1\rangle$$

and we have

$$\begin{aligned}
 & (R_\theta \otimes I_{m+n})(I \otimes U_{\psi_0}) |0\rangle |0^m\rangle |0^n\rangle \\
 &= \cos \theta |0\rangle (\sqrt{p_0} |0^m\rangle |\psi_0\rangle + \sqrt{1-p_0} |\perp'\rangle) + \sin \theta |1\rangle (\sqrt{p_0} |0^m\rangle |\psi_0\rangle + \sqrt{1-p_0} |\perp'\rangle) \\
 &= \sqrt{p_0} \cos \theta |0\rangle |0^m\rangle |\psi_0\rangle + \sqrt{1-p_0 \cos^2 \theta} |\perp'\rangle
 \end{aligned}$$

Here $(|0^{m+1}\rangle \langle 0^{m+1}| \otimes I_n) |\perp'\rangle = 0$. We only need to choose $\sqrt{p_0} \cos \theta = \alpha$.

19.10 Applications of Grover's Algorithm

19.10.1 Application: Satisfiability

Grover's algorithm has many applications: basically any classical algorithm that has some search component can be improved using Grover's algorithm as a subroutine. This includes many basic computer applications such as finding shortest paths and minimum spanning trees, various other graph algorithms, etc..

We can also use it to speed up the computation of NP-complete problems, albeit only quadratically, not exponentially. As an example, consider the satisfiability problem: we are given a Boolean function $\phi(i_1, \dots, i_n)$ and want to know if it has a satisfying assignment, i.e., a setting of the bits i_1, \dots, i_n that makes $\phi(i_1, \dots, i_n) = 1$. A classical brute-force search along all 2^n possible assignments takes time roughly 2^n .

To find a satisfying assignment faster, we define the $N = 2^n$ -bit input to Grover's algorithm by $x_i = \phi(i)$, where $i \in \{0, 1\}^n$. For a given assignment $i = i_1, \dots, i_n$ it is easy to compute $\phi(i)$ classically in polynomial time. We can write that computation as a reversible circuit (using only Toffoli gates), corresponding to a unitary U_ϕ that maps $|i, 0, 0\rangle \rightarrow |i, \phi(i), w_i\rangle$, where the third register holds some classical workspace the computation might have needed. To apply grover we need an oracle that puts the answer in the phase and doesn't leave workspace around (as that could mess up the interference effects). Define $O_{x,\pm} |i\rangle = (-1)^{x_i} |i\rangle$; here we did not explicitly write the workspace qubits, which start and end in $|0\rangle$. Now we can run Grover and find a satisfying assignment with high probability if there is one, using a number of elementary operations that is $\sqrt{2^n}$ times some polynomial factor.

If brute-force search is basically the best thing we can do classically to solve some particular NP-hard problem, then that computation can be sped up quadratically on a quantum computer using Grover search like above. However, there are also NP-hard problems where we know algorithms that still run in exponential time, but that

are much faster than brute-force search. For example, consider the famous Travelling Salesman Problem (TSP): given an n -vertex graph with weights (distances) on the edges, find the shortest tour in this graph that visits every node exactly once. Since there are $(n - 1)!$ many different tours, classical brute-force search would take time $(n - 1)!$ times some polynomial in n . Grover's algorithm could speed this up quadratically. However, there are much more clever classical algorithms for TSP. In particular, Bellman-Held-Karp dynamic programming algorithm solves TSP in time 2^n , times a polynomial in n . This algorithm is much faster than $O(\sqrt{n!})$ (which is roughly $(n/e)^{n/2}$), and is not amenable to a straightforward speed-up using Grover. Nevertheless, it turns out quantum computers can still solve TSP polynomially faster than the best known classical algorithms, albeit in a much more complicated way than by just applying Grover.[?].

19.10.2 Speeding up the solutions of NP-complete problems

Quantum searching may be used to speed up the solution to the problems in the complexity class NP. We already saw, how factoring can be sped up here, we illustrate how quantum search can be applied to assist the solution of the Hamiltonian cycle problem (HC). Recall that a Hamiltonian cycle of a graph is a simple cycle which visits every vertex of the graph. The HC problem is to determine where a given graph has a Hamiltonian cycle or not. This problem belongs to the class of NP-complete problems, widely believed (but not yet proved) to be intractable on a classical computer. A simple algorithm to solve HC is to perform a search through all possible orderings of the vertices:

1. Generate each possible ordering (v_1, \dots, v_n) of vertices for the graph. Repetitions will be allowed, as they ease the analysis without affecting the essential result.
2. For each ordering check whether it is a hamiltonian cycle for the graph. If not, continue checking the orderings.

Since there are $n^n = 2^{n \log n}$ possible orderings of the vertices which must be searched for, this algorithm requires $2^{n \log n}$ checks for the Hamiltonian cycle property in the worst case. Indeed, any problem in NP may be solved in a similar way: if a problem of size n has witnesses which can be specified using $\omega(n)$ bits, where $\omega(n)$ is some polynomial in n , then searching through all $2^{\omega(n)}$ possible witnesses will reveal a solution to the problem, if one exists.

The quantum search algorithm may be used to speed up this algorithm by increasing the speed of the search. Specifically, we use the algorithm described earlier

to determine whether a solution to the search problem exists. Let $m = \lceil \log n \rceil$. The search space for the algorithm will be represented by a string of mn qubits, each block of m qubits being used to store the index to a single vertex. Thus we can write the computational basis states as $|v_1, \dots, v_n\rangle$, where each $|v_i\rangle$ is represented by the appropriate string of m qubits, for a total of nm qubits. the oracle for the search algorithm must apply the transformation:

$$O|v_1, \dots, v_n\rangle = \begin{cases} |v_1, \dots, v_n\rangle, & \text{if } v_1, \dots, v_n \text{ is not a Hamiltonian cycle} \\ -|v_1, \dots, v_n\rangle, & \text{if } v_1, \dots, v_n \text{ is a Hamiltonian cycle} \end{cases}$$

Such an oracle is easy to design and implement when one has a description of the graph. One takes a polynomial size classical circuit recognizing Hamiltonian cycles in the graph and converts it to a reversible circuit, also of polynomial size, computing the transformation $|v_1, \dots, v_n, q\rangle \rightarrow |v_1, \dots, v_n, q \oplus f(v_1, \dots, v_n)\rangle$, where $f(v_1, \dots, v_n) = 1$ if (v_1, \dots, v_n) is a Hamiltonian cycle, and is 0 otherwise. Implementing the corresponding circuit on a quantum circuit with the final qubit starting in the state $(|0\rangle - |1\rangle)/\sqrt{2}$ gives the desired transformation. We won't explicitly describe the details here, except to note the key point: the oracle requires a number of gate as polynomial in n , as a direct consequence of the fact that Hamiltonian cycles can be recognized using polynomially many gates classically. Applying the variant of the search algorithm which determines whether a solution to the search problem exists. We see that it takes $O(2^{mn/2}) = O(2^{n\lceil \log n \rceil / 2})$ applications of the oracle to determine whether a Hamiltonian cycle exists. When one does exist it is easy to apply the combined counting-search algorithm to find an example of such a cycle, which can be exhibited as a witness for the problem.

To summarize:

- The classical algorithm requires $\Theta(p(n)2^{n\lceil \log n \rceil})$ operations to determine whether a Hamiltonian cycle exists, where the polynomial factor $p(n)$ is overhead predominantly due to the implementation of the oracle, that is, the gates checking whether a candidate paths is Hamiltonian or not. The dominant effect in determining the resources requires is the exponent in $2^{n\lceil \log n \rceil}$. The classical algorithm is deterministic, that is, it succeeds with probability 1.
- the quantum algorithm requires $\Theta(p(n)2^{n\lceil \log n \rceil / 2})$ operations to determine whether a Hamiltonian cycle exists. Once again, the polynomial $p(n)$ is overhead predominantly due to implementation of oracle. the dominant effect in determining the resources requires is the exponent in $2^{n\lceil \log n \rceil / 2}$. There is a constant probability (say, $1/6$) of error for the algorithm, which may be reduced to $1/6^r$ by r repetitions of the algorithm.

- Asymptotically the quantum algorithm requires the square root of the number of operations the classical algorithm requires.

19.10.3 Quantum Search of an unstructured database

Suppose somebody gives you a list containing one thousand flower names, and asks you where ‘Peth Rose’ appears on the list. If the flower appears exactly once on the list, and the list is not ordered in any obvious way, then you will need to examining five hundred names, on average, before you find the ‘Peth Rose’. Might it be possible to speed up this kind of database searching using the quantum search algorithm? Indeed, the quantum search algorithm is sometimes referred to as a database search algorithm, but its usefulness for that application is limited, and based on certain assumptions. In this section we take a look at how the quantum search algorithm can conceptually be used to search an unstructured database, in a setting rather like that found on a conventional computer. The picture we construct will clarify what resources are required to enable a quantum computer to search classical databases.

Suppose we have a database containing $N = 2^n$ items, each of length l bits. We will label these items d_1, \dots, d_N . We want to determine where a particular l bit string, s , is in the database. A classical computer, used to solve this problem, is typically split into two parts, illustrated in figure 19.18. One is the central processing unit, or CPU, where data manipulation takes place, using a small amount of temporary memory. The second part is a large memory which stores the database in a string of 2^n blocks of l bit cells. The memory is assumed to be passive, in the sense that it is not capable of processing data on its own. What is possible is to LOAD data from memory into the CPU, and store data from the CPU in memory, and to do manipulations of the data stored temporarily in the CPU. Of course, the classical computers may be designed along different lines, but this CPU-memory split is a popular common architecture. To find out where a given string s is in the unstructured database, the most efficient classical algorithm is as follows. First, an n bit index to the database elements is set up in the CPU. We assume that the CPU is large enough to store the $n = \lceil \log N \rceil$ bit index. The index starts out at zero, and is incremented by one on each iteration of the algorithm. At each iteration, the database entry corresponding to the index is loaded into the CPU, and compared to the string which is being searched for. If they are the same, the algorithm outputs the value of the index and halts. If not, the algorithm continues incrementing the index. Obviously, this algorithm requires that items be loaded from memory 2^n times in the worst case. It is also clear that this is the most efficient possible algorithm for solving the problem in this model of computation.

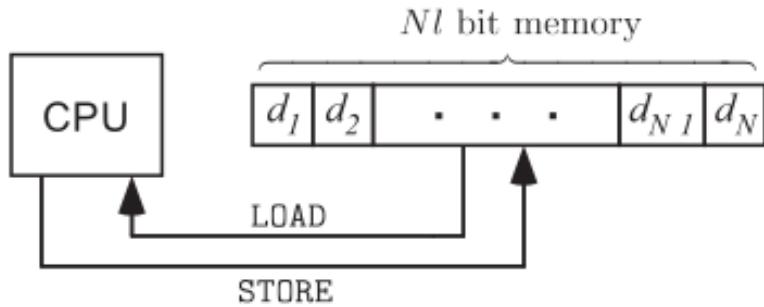


Figure 19.18: Classical Database searching on a computer with distinct central processing unit (CPU) and memory. Only two operations may be directly performed on the memory - a memory element may be LOADED into the CPU, or an item from the CPU may be STOREd in memory

How efficiently can an analogous algorithm be implemented on a quantum computer? And, even if a quantum speedup is possible, how useful is such an algorithm? We show first that a speedup is possible, and then return to the question of the utility of such an algorithm. Suppose our quantum computer consists of two units, just like the classical computer, a CPU and a memory. We assume that the CPU contains four registers (1) an n qubit ‘index’ register initialized to $|0\rangle$; (2) an l qubit register initialized to $|s\rangle$ and remaining in that state for the entire computation; (3) an l qubit ‘data’ register initialized to $|0\rangle$; and (4) a 1 qubit register initialized to $(|0\rangle - |1\rangle)/\sqrt{2}$.

The memory unit can be implemented in one of two ways. The simplest is a quantum memory containing $N = 2^n$ cells of l qubits, each, containing the database entries $|d_x\rangle$. The second implementation is to implement the memory as a classical memory with $N = 2^n$ cells of l bits each, containing the database entries d_x . Unlike a traditional memory, however, it can be addressed by an index x which can be in a superposition of multiple values. This quantum index allows a superposition of cell values to be LOADED from memory. Memory access works in the following way: if the CPU’s index register is in the state $|x\rangle$ and the data register is in the state $|d\rangle$, then the contents d_x of the x th memory cell are added to the data register : $|d\rangle \rightarrow |d \oplus d_x\rangle$, where the addition is done bitwise, modulo 2. First, let us see how this capability is used to perform quantum search, then we shall discuss how such a memory might be physically constructed.

The key part of implementing quantum search algorithm is realization of oracle, which must flip the phase of the index which locates s in the memory. Suppose the

CPU is in the state

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

Applying the LOAD operation puts the computer in the state

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

Now the second and third registers are compared and if they are the same, then a bit flip is applied to register 4; otherwise nothing is changed. The effect of this operation is

$$|x\rangle |s\rangle |d_x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \rightarrow \begin{cases} -|x\rangle |s\rangle |d_x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}, & \text{if } d_x = s \\ |x\rangle |s\rangle |d_x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}, & \text{if } d_x \neq s \end{cases}$$

The data register is then restored to the state $|0\rangle$ by performing the LOAD operation again. The total action of the oracle thus leaves registers 2, 3 and 4 unaffected, and unentangled with register 1 from $|x\rangle$ to $-|x\rangle$ if $d_x = s$, and to leave the register alone otherwise. Using the oracle implemented in this way, we may apply the quantum search algorithm to determine the location of s in the database, using $\mathcal{O}(\sqrt{N})$ LOAD operations, compared to the N LOAD operations required classically.

In order for the oracle to function correctly on superpositions it seems at first glance as though the memory needs to be quantum mechanical. In fact, as we noted above, with some caveats the memory can actually be implemented classically, which likely makes it much more resistant to the effects of noise. But a quantum addressing scheme is still needed; a conceptual picture illustrating how this might be done is shown in figure 19.19. The principle of operation is a means by which the binary encoded state of the quantum index (where 0 to $2^n - 1$ is represented by n qubits) is translated into a unary encoding (where 0 to $2^n - 1$ is represented by the position of a single problem within 2^n possible locations) which address the classical database. The database effects a change on a degree of freedom within the problem which is unrelated to its position. The binary to unary encode is then reversed, leaving the data register with the desired contents.

Are there practical instances in which the quantum search algorithm could be useful for searching classical databases? Two distinct points can be made. First, databases are not ordinarily unstructured. Simple databases, like one containing flower names discussed in the introduction of this section, may be maintained in alphabetical order, such that a binary search can be used to locate an item in time which is $\mathcal{O}(\log(N))$ for an N -element database. However, some databases may require a much more complex structure, and although sophisticated techniques exist

to optimize classical searches, given queries of a sufficiently complex of unanticipated nature, a predetermined structure may not be of assistance, and the problem can be regarded as being essentially the unstructured search problem as we discussed.

Second, for a quantum computer to be able to search classical database, a quantum addressing scheme is required. The scheme we depicted requires $\mathcal{O}(N \log N)$ quantum switches about the same amount of hardware as would be required to store the database itself. Presumably, these switches may some day be as simple and inexpensive as classical memory elements, but if that is not the case, then building a quantum computer to perform quantum search may not be economically advantageous, compared with using classical computing hardware distributed over the memory elements,

Given these considerations, it appears that the principle of quantum search algorithms will not be in searching classical databases. Rather, their use will probably be in searching for solutions to hard problems such as the Hamiltonian cycle, traveling salesman, and satisfiability problems.

19.11 Implementation using Qiskit

19.11.1 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.

19.11.2 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

19.11.3 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
        bits
5
6 Parameters:

```

```

7     marked_states (str or list): Marked states of oracle
8
9 Returns:
10    QuantumCircuit: Quantum circuit representing Grover oracle
11 """
12 if not isinstance(marked_states, list):
13     marked_states = [marked_states]
14 # Compute the number of qubits in circuit
15 num_qubits = len(marked_states[0])
16
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     # where the target bit-string has a '0' entry
28     qc.x(zero_inds)
29     qc.compose(MCMT(ZGate(), num_qubits - 1, 1), inplace=True)
30     qc.x(zero_inds)
31 return qc
32

```

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 19.20 **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 19.21. 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)))

```

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 19.22

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_is_a = pm.run(qc)
7 circuit_is_a.draw(output="mpl", idle_wires=False, style="iqp")

```

This outputs the following 19.23.

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 19.24

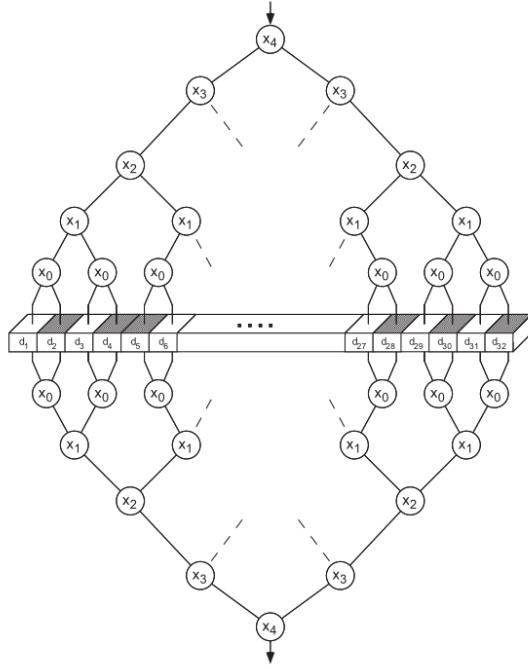


Figure 19.19: Conceptual diagram of a 32 cell classical memory with a five qubit quantum addressing scheme. Each circle represents a switch, addressed by the qubit inscribed within. For example, when $|x_4\rangle = |0\rangle$, the corresponding switch routes the input qubit towards the left; when $|x_4\rangle = |1\rangle$ the switch routes the input qubit to the right. IF $|x_4\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, then an equal superposition of both routes is take. The data register qubits enter at the top of the tree, and are routed down to the database, which changes their state according to the contents of the memory. The qubits are then routed back into a definite position, leaving them with the retrieved information. Physically, this could be realized using, for example, single photons, for the data register qubits, which are steered using nonlinear interferometers. The classical database could be just a simple sheet of plastic in which a ‘zero’ (illustrated as white squares) transmits light unchanged, and a ‘one’ (shaded squares) shifts polarization of the incident light by 90°

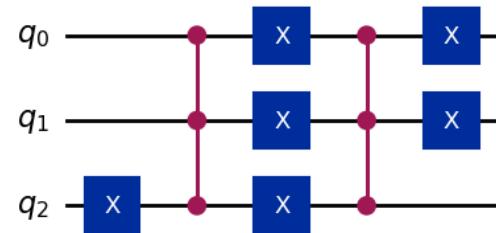


Figure 19.20: Specific Grover's Instance

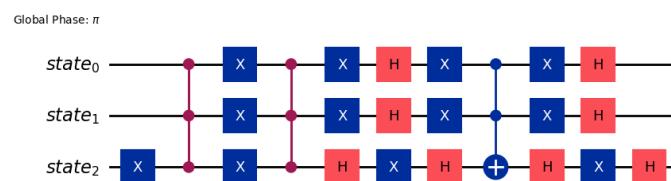


Figure 19.21: Grover Operator

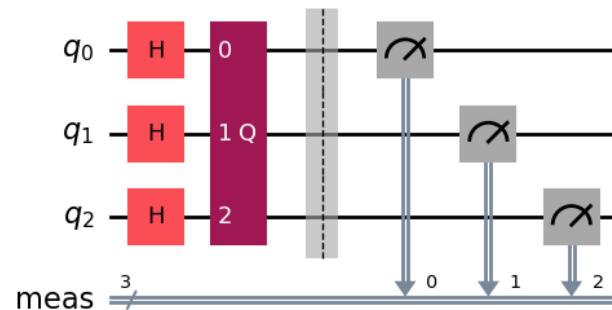


Figure 19.22: Full Grover Circuit

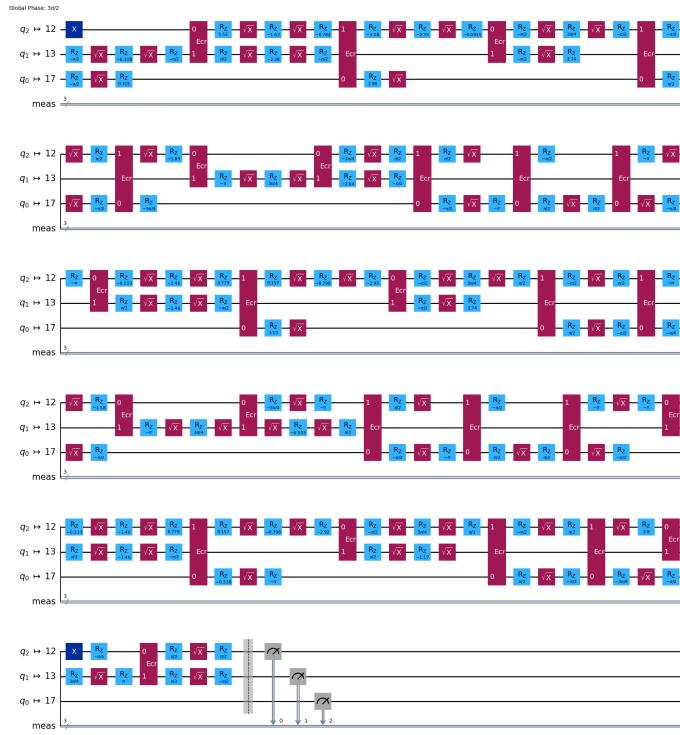


Figure 19.23: Optimized Circuit

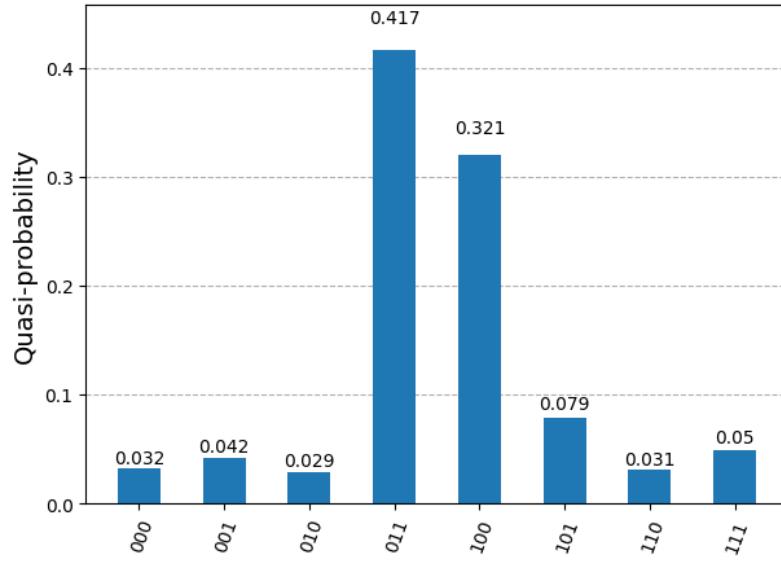


Figure 19.24: Probability plot

Chapter 20

Trotter based Hamiltonian Simulation

Thus, far we have viewed the dynamics of quantum systems from the perspective of unitary transformations: apart from measurement, the only way a quantum state (i.e. a vector of amplitudes) can change is by multiplication with a unitary matrix, for instance a 2-qubit gate tensored with identities on other qubits. but which unitary will actually occur in a given physical system? This is determined by the Hamiltonian of the system, which is the observable H corresponding to the total energy in the system. The expectation value $\langle \psi | H | \psi \rangle$. Typically, this total energy is the sum of several different terms, corresponding to kinetic energy, potential energy etc. Also typically, it is the sum of many local terms that each act on only a few of the particles (qubits) of the system, for example if all interaction are between pairs of particles. One can think of the Hamiltonian H as describing the physical characteristics of the system. They do determine the evolution of the state in time i.e. the state $|\psi(t)\rangle$ as a function of the time parameter t , given an initial state $|\psi(0)\rangle$.

Trotterization in quantum mechanics is an important theoretical concepts in handling the exponential of non commutative operators. In this communication, we give a mathematical formulation of the trotter Product formula, and apply it to basic examples in which the utility of trotterization is evident.

Proposition 20.0.1. (*Duhamel's principle for Hamiltonian Simulation*) Let $U(t), \tilde{U}(t) \in \mathbb{C}^{N \times N}$ satisfy

$$\imath \partial_t U(t) = HU(t), \quad \imath \partial_t \tilde{U}(t) = H\tilde{U}(t) + B(t), \quad U(0) = \tilde{U}(0) = I$$

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

20.1 Simulation of Quantum Systems

A useful practical application of quantum circuit model is computation of physical systems. For example, finite element analysis and modelling of bridges to ensure safety while minimizing cost. Cars are made lightweight, structurally sound, attractive, and inexpensive, by using computer aided design. Modern aeronautical engineering depends heavily on computational fluid dynamics simulation for aircraft designs. Nuclear weapons are no longer exploded (for the most part), but rather, tested by exhaustive computational modeling. Examples abound, because of the tremendous practical applications of predictive simulations.

20.1.1 Simulation in action

The heart of any simulation is solving differential equations which capture the physical laws governing the dynamical behavior of a system. For example, Newton's Law

$$\frac{d}{dt} \left(m \frac{dx}{dt} \right) = F,$$

Poisson's equation,

$$-\nabla \cdot (k \nabla \vec{u}) = \vec{Q}$$

the electromagnetic vector wave equation,

$$\nabla \cdot \nabla \vec{E} = \epsilon_0 \mu_0 \frac{\partial^2 \vec{E}}{\partial t^2}$$

and the diffusion equation

$$\vec{\nabla}^2 \psi = \frac{1}{\alpha^2} \frac{\partial \psi}{\partial t}$$

The goal is generally: given an initial state of the system, what is the state at some other time and/or position. Solutions are generally obtained by approximating the state with a digital representation, then discretizing the differential equation in space and time such that an iterative application of a procedure carries the state from the initial to the final conditions. Importantly, the error in this procedure is bounded, and known not to grow faster than some small power of the number of iterations.

Furthermore, not all dynamical systems be simulated efficiently: generally only those systems which can be described efficiently can be simulated efficiently.

Simulation of quantum systems by classical computers is possible, but generally only very inefficiently. The dynamical behavior of many simple quantum systems is governed by Schrodinger's equation,

$$\imath \hbar \frac{d}{dt} |\psi\rangle = H |\psi\rangle$$

We will find it convenient to absorb \hbar into H , and use this convention for the rest of this section. For a typical Hamiltonian of interest to physicists deal with real particles in space (rather than abstract systems such as qubits, which we have been dealing with) this reduces to

$$\imath \frac{\partial}{\partial t} \psi(x) = \left(-\frac{1}{2m} \frac{\partial^2}{\partial x^2} + V(x) \right) \psi(x)$$

using a convention known as the position representation $\langle x|\psi\rangle = \psi(x)$. This is also an elliptical equation. So just simulating Schrodinger's equation is not the especial difficulty faced in simulation quantum systems. The key challenge in simulating quantum systems is the exponential number of differential equations which must be solved. For one qubit evolving according to the Schrodinger equation, a system of two differential equations must be solved; for two qubits, four equations; and for n qubits 2^n equations. Sometimes' insightful approximations can be made which reduce the effective number of equations involved, thus making classical simulation of the quantum system feasible. However there are many physically interesting quantum systems for which no such approximations are known.

Example 20.1.1. (Exponential complexity growth of quantum systems)
Let ρ be a density matrix describing the state of n qubits. Show that describing ρ requires $4^n - 1$ independent real numbers.

An operator ρ is the density operator associated to some ensemble $\{p_i, |\psi_i\rangle\}$ if and only if it satisfies the conditions:

1. **(Trace condition):** ρ has trace equal to one.
2. **(Positivity condition)** ρ is a positive operator.

Note that ρ will be a $2^n \times 2^n$ dimension matrix. For ρ to be a positive operator it must be Hermitian matrix and for a Hermitian matrix the diagonal elements will always be real. Thus the total number of complex values required is $1 + 2 + 3 + \dots + 2^n - 1 = \frac{2^n(2^n - 1)}{2} = \frac{4^n - 2^n}{2}$. Hence the number of real values required will be $4^n - 2^n$ (2 for each complex numbers). There are 2^n real numbers with the relation that the trace has to be equal to one. Thus, $2^n - 1$ independent real numbers for the diagonal. Thus, in total we require $4^n - 2^n + 2^n - 1 = 4^n - 1$ independent real numbers.

The reader with a physics background may appreciate that there are many important quantum systems for which classical simulation is intractable. These include the Hubbard model, a model of interacting fermionic particles with the Hamiltonian

$$H = \sum_{k=1}^n V_0 n_{k\uparrow} n_{k\downarrow} + \sum_{k,j,\text{neighbors},m,\sigma} t_0 c_{k\sigma}^* c_{j\sigma}$$

which is useful in the study of superconductivity and magneitsm, the Ising model,

$$H = \sum_{k=1}^n \vec{\sigma}_k \cdot \vec{\sigma}_{k+1}$$

and many others. Solutions to such models give many physical properties such as the dielectric constant, conductivity, and magnetic susceptibility of materials. More sophisticated models such as quantum electrodynamics (QED) and quantum chromodynamics (QCD) can be used to compute constants such as the mass of the proton.

Quantum computers can efficiently simulate quantum systems for which there is no known efficient classical simulation. Intuitively, this is possible for the same reason any circuit can be constructed from a universal set of quantum gates. Moreover, just as there exist unitary operations which cannot be efficiently approximated, it is possible in principle to imagine quantum systems which cannot be efficiently simulated on a quantum computer. Of course, we believe that such systems aren't actually realized in Nature, otherwise we'd be able to exploit them to do information processing beyond the quantum circuit model.

20.2 Introduction

One of the major goals of quantum mechanics is finding solutions, called wavefunctions/eigenfunctions, to the time-independent Schrodinger wave equation. For a given time-independent Hamiltonian operator \hat{H} on a Hilbert space \mathcal{H} , the Schrodinger equation is given by

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

where $|\psi\rangle \in \mathcal{H}$ is an eigenfunction and $E \in \mathbb{R}$ is an energy eigenvalue. Alone, this equation only yields the energy values and the stationary states of the physical system. To get the time-dependent eigenvector one needs the unitary operator $U(t)$:

$$U(t) = e^{-\imath \hat{H} t / \hbar}$$

$$|\psi(t)\rangle = U(t) |\psi\rangle = e^{-\imath \hat{H} t / \hbar} |\psi\rangle$$

where $\hbar = \frac{h}{2\pi}$ is Planck's reduced constant.

In practice, it can be hard to compute this operator exponential, so let us focus on the simplest example: When the (finite-dimensional in this case) Hamiltonian operator \hat{H} is given by the diagonal matrix

$$\hat{H} = \begin{pmatrix} E_1 & 0 & \dots & 0 \\ 0 & E_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & E_n \end{pmatrix}$$

Substituting this in to equation, the eigenfunctions are given by the standard basis

$$|\psi_1\rangle = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, |\psi_2\rangle = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \dots, |\psi_n\rangle = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

with corresponding eigenvalues E_1, \dots, E_n , respectively. From this, an arbitrary stationary state $|\psi\rangle$ of the entire quantum state is simply a complex ($c_i \in \mathbb{C}$) linear combination:

$$|\psi\rangle = \sum_{i=1}^n c_i |\psi_i\rangle, \sum_{i=1}^n |c_i|^2 = 1$$

This is the time-independent solution. In general H may itself change with t , but for simplicity we will only consider here the case where H is time-independent. To get the solution $|\psi(t)\rangle$, we simply need to compute $U(t)$. This is made easy by the fact that the exponential of a diagonal matrix is just the diagonal matrix of element exponentials. Concretely,

$$\begin{aligned} U(t) &= \exp \begin{pmatrix} -\imath E_1 t / \hbar & 0 & \dots & 0 \\ 0 & -\imath E_2 t / \hbar & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -\imath E_n t / \hbar \end{pmatrix} \\ &= \begin{pmatrix} e^{-\imath E_1 t / \hbar} & 0 & \dots & 0 \\ 0 & e^{-\imath E_2 t / \hbar} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & e^{-\imath E_n t / \hbar} \end{pmatrix} \end{aligned}$$

Therefore, the time-dependent state $|\psi(t)\rangle$ is give by

$$\begin{aligned} |\psi(t)\rangle &= U(t) |\psi\rangle \\ &= \begin{pmatrix} e^{-\imath E_1 t / \hbar} & 0 & \dots & 0 \\ 0 & e^{-\imath E_2 t / \hbar} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & e^{-\imath E_n t / \hbar} \end{pmatrix} \left(\sum_{i=1}^n c_i |\psi_i\rangle \right) \\ &= \sum_{i=1}^n c_i e^{-\imath E_i t / \hbar} |\psi_i\rangle \end{aligned}$$

This calculation was simple because the Hamiltonian \hat{H} was diagonal, but for the general case we need to use the series definition of exponentiation. Given a (potential infinite dimensional) self-adjoint (Hermitian) operator S defined on \mathcal{H} , its exponential is defined by

$$e^S = \sum_{n=0}^{\infty} \frac{S^n}{n!}$$

When S is a finite-dimensional simply diagonalizing the matrix is sufficient to compute e^S , and therefore solve the eigenvalue problem. However, when S is infinite dimensional, the problem becomes much more complicated. To help reduce this complexity, we are considering decomposing an operator into a sum.

If we have another self adjoint operator T defined on \mathcal{H} such that $[S, T] = 0$ (*i.e.* $ST = TS$), then the exponential of the sum splits:

$$e^{S+T}\xi = e^S e^T \xi \quad \forall \xi \in \mathcal{H}$$

We can actually prove this in a relatively straight-forward manner from the definition of the operator exponential by applying the binomial theorem:

$$\begin{aligned} e^{S+T} &= \sum_{n=0}^{\infty} \frac{(S+T)^n}{n!} \\ &= \sum_{n=0}^{\infty} \sum_{k=0}^n \frac{1}{n!} C_k S^{n-k} T^k \\ &= \sum_{n=0}^{\infty} \sum_{k=0}^n \frac{1}{n!} \frac{n!}{(n-k)!k!} S^{n-k} T^k \\ &= \sum_{n=0}^{\infty} \sum_{k=0}^n \frac{1}{(n-k)!k!} S^{n-k} T^k \end{aligned}$$

With this result, notice that every possible product of S^m with T^n occurs for $m, n \in \mathbb{Z}^+ \cap \{0\}$. Thus, rewrite the sum as follows:

$$\begin{aligned} \sum_{n=0}^{\infty} \sum_{k=0}^n \frac{1}{(n-k)!k!} S^{n-k} T^k &= \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} \frac{1}{m!n!} S^m T^n \\ &= \left(\sum_{m=0}^{\infty} \frac{1}{m!} S^m \right) \left(\sum_{n=0}^{\infty} \frac{1}{n!} T^n \right) \\ &= e^S e^T \end{aligned}$$

In the case where S and T do not commute, this argument fails because the Binomial Theorem no longer applies. It seems like there is no way to generalize this argument for the noncommutative case. For example, in the binomial expansion of $(S+T)^3$ with $[S, T] \neq 0$, $STS \neq S^2T$, and so it's impossible to collect terms on the left and right sides overall sum. A somewhat surprising result called the Trotter Product Formula is needed for noncommutative operators, which in a strong sense approximates the above splitting of the exponential. This is what we explore for the remainder of this section.

Classical simulation begins with the realization that in solving a simple differential equation such as $dy/dt = f(y)$, to first order, it is known that $y(t + \Delta t) \approx y(t) + f(y)\Delta t$. Similarly, the quantum case is concerned with the solution of $\iota \frac{d|\psi\rangle}{dt} = H |\psi\rangle$, which for a time-independent H , is just

$$|\psi(t)\rangle = e^{-\iota Ht} |\psi(0)\rangle$$

Since H is usually extremely difficult to exponentiate (it may be sparse, but it is also exponentially large), a good beginning is the first order solution. Note that,

$$\begin{aligned} |\psi(t_1)\rangle &= e^{-\iota Ht_1} |\psi(0)\rangle \\ |\psi(t_2)\rangle &= e^{-\iota Ht_2} |\psi(0)\rangle \end{aligned}$$

Using the fact that $e^{-\iota H(t_2-t_1)} = e^{-\iota Ht_2 + \iota Ht_1} = e^{-\iota Ht_2} e^{\iota Ht_1}$ as decomposition since the matrices $(-\iota Ht_2)(\iota Ht_1) = (\iota Ht_2)(-\iota Ht_1) = H^2 t_1 t_2$, we can thus write

$$\begin{aligned} |\psi(t_2)\rangle &= e^{-\iota Ht_2} |\psi(0)\rangle \\ |\psi(t_2)\rangle &= e^{-\iota H(t_2)} e^{\iota Ht_1} e^{-\iota Ht_1} |\psi(t_1)\rangle \\ |\psi(t_2)\rangle &= e^{-\iota H(t_2-t_1)} e^{-\iota Ht_1} |\psi(0)\rangle \\ |\psi(t_2)\rangle &= e^{-\iota H(t_2-t_1)} |\psi(t_1)\rangle \end{aligned}$$

Now, putting $t_1 = t$ and $t_2 = t_1 + \Delta t$, we get,

$$|\psi(t + \Delta t)\rangle = e^{-\iota H\Delta t} |\psi(t)\rangle$$

Thus, $|\psi(t + \Delta t)\rangle \approx (I - \iota H\Delta t) |\psi(t)\rangle$ which roots from the fact that $|\psi(t_2)\rangle = e^{-\iota H(t_2-t_1)} |\psi(t_1)\rangle = e^{-\iota H\Delta t} |\psi(t_1)\rangle$ by putting $t_1 = t$ and $t_2 = t_1 + \Delta t$ and using the fact that (since the matrices Ht_1 and Ht_2 commute) we can decompose them as $e^{-\iota H(t_2-t_1)} = e^{-\iota Ht_2} e^{\iota Ht_1}$. We get,

$$|\psi(t + \Delta t)\rangle \approx (I - \iota H\Delta t) |\psi(t)\rangle$$

However, such first order solutions are generally not very satisfactory.

Efficient approximation for the solution to high order, is possible for many classes of Hamiltonians. For example, in most physical systems, the hamiltonian can be written as a sum over many local interactions. Specifically for a system for n particles,

$$H = \sum_{k=1}^L H_k$$

wherever each H_k acts on at most a constant c number of systems, and L is polynomial in n . For example, the terms H_k are often just two-body interactions such as $X_i X_j$ and one body Hamiltonians such as X_i . Both the Hubbard and Ising models have Hamiltonians of this form. Such locality is quite physically reasonable, and originates in many systems from the fact that the most interactions fall off with increasing distance or difference in energy. There are sometimes additional global symmetry constricts such as particle statistics. The important point is that although $e^{-\iota Ht}$ is difficult to compute, $e^{-\iota H_k t}$ act on a much smaller subsystem, and is straightforward to approximate using quantum circuits. But because $[H_j, H_k] \neq 0$ in general, $e^{-\iota Ht} \neq \prod_k e^{-\iota H_k t}!$ How, then can $e^{-\iota H_k t}$ be useful in constructing $e^{-\iota Ht}$. Now, we shall also see the extension of this proof where H can be decomposed onto a sum of L Hamiltonians such that they commute. In other words

Theorem 20.2.1. *Show that $e^{-\iota \hat{H}t} = e^{-\iota \hat{H}_1 t} e^{-\iota \hat{H}_2 t} \dots e^{-\iota \hat{H}_L t}$ if $[\hat{H}_i, \hat{H}_j] = 0$ where each $\hat{H}_i = \hat{H}_i^\dagger$ i.e. are Hermitian.*

Proof. For any local Hamiltonian $\hat{H} = \sum_{l=1}^L \hat{H}_l$ which consists of L Hermitian terms $\hat{H}_l = \hat{H}_l^\dagger$, we are required to prove that the evolution under these Hamiltonian can be decomposed into the product of evolution of local terms if only these terms commute with each other. For

$$\hat{H} = \sum_{l=1}^L \hat{H}_l$$

we are required to prove that $e^{-\iota \hat{H}t} = e^{-\iota \hat{H}_1 t} e^{-\iota \hat{H}_2 t} \dots e^{-\iota \hat{H}_L t}$. Without the loss of generality we can prove the above statement for the Hamiltonian consisting of two local terms i.e. $e^{-\iota(\hat{H}_1 + \hat{H}_2)t} = e^{-\iota \hat{H}_1 t} e^{-\iota \hat{H}_2 t}$, where $\hat{H} = \hat{H}_1 + \hat{H}_2$. Using the formula for Maclaurian series

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Thus,

$$e^{-\iota(\hat{H}_1 + \hat{H}_2)t} = \sum_{n=0}^{\infty} \frac{(-\iota t \hat{H}_1 - \iota t \hat{H}_2)^n}{n!}$$

Then applying the Binomial formula $(x+y)^n = \sum_{k=0}^n \frac{n!}{k!(n-k)!} x^{n-k} y^k$. Note that if $\hat{H}_1 \hat{H}_2 \neq \hat{H}_2 \hat{H}_1$ then in $(\hat{H}_1 + \hat{H}_2)^3 \Rightarrow \hat{H}_1^2 \hat{H}_2 \neq \hat{H}_1 \hat{H}_2 \hat{H}_1$, i.e. the case when \hat{H}_1 and \hat{H}_2 do not commute, the Binomial formula is not applicable as reasoned and it is impossible to group them. For our proof since we assume that they commute and using the Binomial formula we get,

$$= \sum_{n=0}^{\infty} \frac{1}{n!} \sum_{k=0}^n \frac{n!}{k!(n-k)!} (-\iota t \hat{H}_1)^{n-k} (-\iota t \hat{H}_2)^k$$

Upon simplifying we get,

$$= \sum_{n=0}^{\infty} \sum_{k=0}^n \frac{(-\iota t \hat{H}_1)^{n-k}}{(n-k)!} \frac{(-\iota t \hat{H}_2)^k}{k!}$$

Note that the inner sum for each n are as follows

$$\begin{aligned} n = 0 &\implies \frac{(-\iota t \hat{H}_1)^0}{0!} \frac{(-\iota t \hat{H}_2)^0}{0!} \\ n = 1 &\implies \frac{(-\iota t \hat{H}_1)^1}{1!} \frac{(-\iota t \hat{H}_2)^0}{0!} + \frac{(-\iota t \hat{H}_1)^0}{0!} \frac{(-\iota t \hat{H}_2)^1}{1!} \\ n = 2 &\implies \frac{(-\iota t \hat{H}_1)^2}{2!} \frac{(-\iota t \hat{H}_2)^0}{0!} + \frac{(-\iota t \hat{H}_1)^1}{1!} \frac{(-\iota t \hat{H}_2)^1}{1!} + \frac{(-\iota t \hat{H}_1)^0}{0!} \frac{(-\iota t \hat{H}_2)^2}{2!} \\ &\quad \frac{(-\iota t \hat{H}_2)^0}{0!} \left(\frac{(-\iota t \hat{H}_1)^0}{0!} + \frac{(-\iota t \hat{H}_1)^1}{1!} + \frac{(-\iota t \hat{H}_1)^2}{2!} + \dots \right) + \\ &\quad \frac{(-\iota t \hat{H}_1)^1}{1!} \left(\frac{(-\iota t \hat{H}_1)^0}{0!} + \frac{(-\iota t \hat{H}_1)^1}{1!} + \frac{(-\iota t \hat{H}_1)^2}{2!} + \dots \right) + \\ &\quad \frac{(-\iota t \hat{H}_2)^2}{2!} \left(\frac{(-\iota t \hat{H}_1)^0}{0!} + \frac{(-\iota t \hat{H}_1)^1}{1!} + \frac{(-\iota t \hat{H}_1)^2}{2!} + \dots \right) + \dots \end{aligned}$$

This expression can thus be further simplifies as a multiplication of two independent sums as

$$= \sum_{q=0}^{\infty} \sum_{r=0}^{\infty} \frac{(-\iota t \hat{H}_1)^q}{q!} \frac{(-\iota t \hat{H}_2)^r}{r!} = \left(\sum_{q=0}^{\infty} \frac{(-\iota t \hat{H}_1)^q}{q!} \right) \left(\sum_{r=0}^{\infty} \frac{(-\iota t \hat{H}_2)^r}{r!} \right) = e^{-\iota \hat{H}_1 t} e^{-\iota \hat{H}_2 t}$$

This can be inductively extended to all the other terms. Thus,

$$e^{-\iota \hat{H}t} = e^{-\iota \hat{H}_1 t} e^{-\iota \hat{H}_2 t} \dots e^{-\iota \hat{H}_L t}$$

if $[\hat{H}_i, \hat{H}_j] = 0$ for all i, j where each $\hat{H}_i = \hat{H}_i^\dagger$ i.e. are Hermitian. \square

Example 20.2.1. Show that the restriction of H_k to involve at most c particles implies that in the sum,

$$H = \sum_{k=1}^L H_k$$

L is upper bounded by a polynomial in n .

If H_k is restricted to involve at most c particles then the number of such Hamiltonians possible will be

$${}^n C_c = \frac{n(n-1)\dots(n-c+1)}{c!} = \mathcal{O}\left(\frac{n^c}{c!}\right)$$

Thus, L will always be upper bounded by a polynomial in n as $\mathcal{O}(n^c)$.

20.3 Hamiltonian Simulation

Quantum systems are described by Hamiltonians where \hat{H} = Kinetic Energy + Potential Energy. For n qubits \hat{H} is of dimension $2^n \times 2^n$ and $|\psi\rangle$ is of dimension $2^n \times 1$. Thus, the Schrodinger equation

$$\iota \hbar \frac{\partial}{\partial t} |\psi\rangle = \hat{H}(t) |\psi\rangle$$

is exponential to solve on a classical computer. Thus, as Feynman said Simulating Physics with Computers and Quantum computers are Feynman's universal quantum simulators (Seth Lloyd). They can calculate evolution of Quantum System Efficiently by doing it in Polynomial time and memory resources in size of target systems. For example, 50 qubits = 2^{50} complex numbers. Even if we consider a float 32 bit precision for each part i.e. 32 bits or Real part and 32 bits for Imaginary part for each complex number. We would still have 8×2^{50} bytes which is equal to $\approx 9PB$ of memory equivalent to modern day RAM of supercomputers.

Simulating dynamics of Quantum systems such as Ising model, Hubbard Model is one of the practical applications of Quantum Computers. Solutions to these models gives many bulk properties such as dielectric constants, connectivity, magnetic

susceptibility of materials and many more. For simulation of atoms, molecules and other biochemical systems we generally require calculation of ground state energy of large systems, study of dynamical behavior of an ensemble of molecules and complex behaviour such as protein folding.

In areas like quantum chemistry (i.e. ,the study of properties of molecules and their interaction) and material sciences, it is often important to figure out how a quantum system will evolve from some given initial state, for instance a basis state,. It is also very important in chemistry to be able to find out global properties of a given hamiltonian like its lowest energy, a.k.a, ground state energy. Unfortunately this problem seems to be hard to solve (in fact it is so-called QMA hard) even for a quantum computer, even for the special case of 2-local Hamiltonians. This is typically hard to do on a classical computers, since the number of parameters (amplitudes) is exponential in the number of particles. However, a quantum computer is like a universal quantum system, and should be able to efficiently simulate every efficient quantum process, in the same way that a classical Turing machine can efficiently simulate other (classical) physical processes. We will see that it is actually possible to classical simulate quantum computers (and hence quantum systems more generally) with a polynomial amount of space, but our best methods still use an exponential amount of time. If factoring a large integer is a hard problem for classical computers (which is widely believed), then Shor's efficient quantum factoring algorithm implies that it is impossible to simulate a quantum computer in polynomial time on a classical computer. In fact, this was the main reason why Feynman invented quantum computers: a controllable quantum system that can be used to simulate other quantum systems. In order to realize that idea, we need methods to efficiently implement the unitary evolution that is induced by a given Hamiltonian. In other words, we need methods to implement $U = e^{-iHt}$ as a quantum circuit of gates (say, up to some small error ϵ), and to apply this to a given initial state $|\psi\rangle$. This is known as the problem of "Hamiltonian Simulation". If n-qubit Unitary \tilde{U} (e.g. a quantum circuit with not too many gates) is meant to approximate n-qubit unitary U , then we can measure the error by the operator norm of their difference $\|U - \tilde{U}\| = \max_{\psi} \|U|\psi\rangle - \tilde{U}|\psi\rangle\|$. However, we will also see simulation methods that allow some auxiliary qubits, say a of them, which start in the state $|0^a\rangle$ and should end in something close to state $|0^a\rangle$. In this case \tilde{U} acts on more qubits than U , so we cannot use the operator norm of their difference, instead we can measure the error on the subspace of $(n + a)$ qubit states where the last a qubits are $|0\rangle$: $\max_{\psi} \|(U|\psi\rangle)|0^a\rangle - \tilde{U}(|\psi\rangle)|0^a\rangle\|$. This way of measuring error still allows you to analyze a sequence of approximate unitaries using triangle inequality in a way that errors add up at most linearly.

20.3.1 Evolution of a Closed Quantum System

Consider the Time-dependent Schrodinger Equation

$$\imath\hbar \frac{\partial}{\partial t} |\psi\rangle = \hat{H}(t) |\psi\rangle$$

where \hbar is the reduced Planck's constant. When the Hamiltonian \hat{H} is constant (does not depend on time) we call it Time independent Hamiltonian and in that case the solution of the Schrodinger's equation is given as

$$|\psi(t)\rangle = e^{-\imath\hat{H}t} |\psi(0)\rangle$$

From this, given a time, we can calculate the state of the system by exponentiating the Hamiltonian. It is generally difficult to exponentiate a Hamiltonian as it may be sparse but exponentially large. Note that $e^{-\imath\hat{H}t}$ is a Unitary matrix.

20.4 The Problem Definition

The Hamiltonian simulation problem with a time-independent Hamiltonian, or the Hamiltonian simulation problem for short is the following problem:

Definition 20.4.1. Given an initial state $|\psi_0\rangle$ and a Hamiltonian H , evaluate the quantum state at time t according to $|\phi(t)\rangle = e^{\imath t H} |\psi_0\rangle$.

Thus, the goal is to find a sequence of computational gates that implement time evolution such that the spectral norm

$$\|U - e^{-\imath\hat{H}t}\|_2 \leq \epsilon$$

recall that $\|A\|_2 = \sqrt{\lambda_{max}(A^\dagger A)} = \sigma_{max}$.

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).

20.5 Trotter Splitting

It is generally difficult to exponentiate a Hamiltonian. In a lot of models, the Hamiltonian can be written as a liner combination of local terms (each term acts only on a part of the total system).

$$\hat{H} = \sum_{l=1}^L \hat{H}_l$$

A k-local Hamiltonian is defined as a Hamiltonian \hat{H} acting on n qubits such that each \hat{H}_l acts non-trivially on atmost k qubits ($k \leq n$) instead of n qubits. Most Hamiltonains that occur in nature are k-local. Assuming that each local term \hat{H}_l is Hermitian i.e. $\hat{H}_l^\dagger = \hat{H}_l$ and can be directly exponentiated. We can present its evolution as a quantum circuit and run it on a quantum computer as

$$e^{-\imath \hat{H}t} = e^{-\imath \hat{H}_1 t} e^{-\imath \hat{H}_2 t} \dots e^{-\imath \hat{H}_L t}$$

if and only if they commute $[H_i, H_j] = 0$ for all i, j . The commuting makes it possible to decompose the evolution of the total Hamiltonian into the product of the local term evolutions. Since each exponent can be represented as a quantum circuit, the Hamiltonian simulation problem is solved for the whole system.

In general case when local terms do not commute Trotterization is used. 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 Δt . 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

$$e^{-\imath tH} = \lim_{L \rightarrow \infty} \left(e^{-\imath \frac{t}{L} H_1} e^{-\imath \frac{t}{L} H_2} \right)^L$$

When taking L to be a finite number, and let $\Delta t = t/L$, this gives the simplest first order Trotter method with

$$\|e^{-\imath \Delta t H} - e^{-\imath \Delta t H_1} e^{-\imath \Delta t H_2}\| = \mathcal{O}(\Delta t^2)$$

Trotter product formula gives a limit when exponentiation is possible even for not commuting parts. This is represented by many names in the literature as Lie-Product formula, Product formula, Trotter's formula, Trotter product formula, Trotter-Suzuki decomposition, Suzuki-Trotter expansion, Splitting method. It was actually given by Sophus Lie (1875), the later on rediscovered by Hale trotter (1959) and further used by Masuo Suzuki (1976). The formula is

$$e^{A+B} = \lim_{n \rightarrow \infty} \left(e^{\frac{A}{n}} e^{\frac{B}{n}} \right)^n$$

for unbounded linear operators A, B . This is called the Lie-Trotter formula. The proof is as follows:

Proof. Let A, B be Hamiltonian operators and $t \in \mathbb{R}$. Recall

$$\begin{aligned} e^x &= \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \\ e^{\iota A t/n} &= I + \frac{1}{n} \iota At + \mathcal{O}\left(\frac{1}{n^2}\right) \\ \implies e^{\iota At/n} e^{\iota Bt/n} &= \left(I + \frac{\iota At}{n}\right) \left(I + \frac{\iota Bt}{n}\right) + \mathcal{O}\left(\frac{1}{n^2}\right) \\ &= I + \frac{\iota At}{n} + \frac{\iota Bt}{n} - \frac{ABt}{n^2} + \mathcal{O}\left(\frac{1}{n^2}\right) \\ &= I + \frac{1}{n} \iota(A+B)t + \mathcal{O}\left(\frac{1}{n^2}\right) \\ \implies (e^{\iota At/n} e^{\iota Bt/n})^n &= I + \sum_{k=1}^n \frac{n!}{(n-k)!k!} \frac{1}{n^k} [\iota(A+B)t]^k + \mathcal{O}\left(\frac{1}{n}\right) \end{aligned}$$

Using the Binomial formula for $(x+y)^n = \sum_{k=0}^n \frac{n!}{(n-k)!k!} x^{n-k} y^k$

$$= \sum_{k=0}^n = \frac{n!}{(n-k)!k!} \frac{1}{n^k} (\iota(A+B)t)^k$$

Now,

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{n!}{(n-k)!k!} \frac{1}{n^k} &= \lim_{n \rightarrow \infty} \frac{1}{k!} \frac{n(n-1)\dots(n-k-1)}{n^k} \\ &= \lim_{n \rightarrow \infty} \frac{1}{k!} \left(1 - \frac{1}{n}\right) \left(1 - \frac{2}{n}\right) \dots \left(1 - \frac{k-1}{n}\right) \\ &\approx \frac{1}{k!} \left(1 + \mathcal{O}\left(\frac{1}{n}\right)\right) \end{aligned}$$

Hence, upon substituting the result in the given equation, we get,

$$\begin{aligned} \lim_{n \rightarrow \infty} (e^{\iota At/n} e^{\iota Bt/n})^n &= \lim_{n \rightarrow \infty} \sum_{k=0}^n \frac{1}{k!} \left(1 + \mathcal{O}\left(\frac{1}{n}\right)\right) (\iota(A+B)t)^k + \mathcal{O}\left(\frac{1}{n}\right) \\ \lim_{n \rightarrow \infty} (e^{\iota At/n} e^{\iota Bt/n})^n &= \sum_{k=0}^n \frac{(\iota(A+B)t)^k}{k!} \\ \lim_{n \rightarrow \infty} (e^{\iota At/n} e^{\iota Bt/n})^n &= e^{\iota(A+B)t} \end{aligned}$$

□

Modifications of the trotter formula provide the methods by which higher order approximations can be derived for performing quantum simulations. For example, using similar reasoning to the proof above, it can be shown that

$$e^{\iota(A+B)\Delta t} = e^{\iota A\Delta t} e^{\iota B\Delta t} + \mathcal{O}(\Delta t^2)$$

Similarly,

$$e^{\iota(A+B)\Delta t} = e^{\iota A\Delta t/2} e^{\iota B\Delta t} e^{\iota A\Delta t/2} + \mathcal{O}(\Delta t^3)$$

Example 20.5.1. Let $H = \sum_k^L H_k$, and define

$$U_{\Delta t} = (e^{-\iota H_1 \Delta t} e^{-\iota H_2 \Delta t} \dots e^{-\iota H_L \Delta t}) (e^{-\iota H_L \Delta t} e^{-\iota H_{L-1} \Delta t} \dots e^{-\iota H_1 \Delta t})$$

- (a) Prove that $U_{\Delta t} = e^{-2\iota H \Delta t} + \mathcal{O}(\Delta t^3)$
- (b) Use the results in Box 4.1 to prove that for a positive integer m ,

$$E(U_{\Delta t}^m, e^{-2m\iota H \Delta t}) \leq m\alpha \Delta t^3$$

(a) Using the following formula,

$$e^{\iota(A+B)\Delta t} = e^{\iota A \Delta t / 2} e^{\iota B \Delta t} e^{\iota A \Delta t / 2} + \mathcal{O}(\Delta t^3)$$

We can then simplify the above expression as follows:

$$\begin{aligned} U_{\Delta t} &= (e^{-\iota H_1 \Delta t} e^{-\iota H_2 \Delta t} \dots e^{-\iota H_L \Delta t}) (e^{-\iota H_L \Delta t} e^{-\iota H_{L-1} \Delta t} \dots e^{-\iota H_1 \Delta t}) \\ &= (e^{-\iota H_1 \Delta t} e^{-\iota H_2 \Delta t} \dots e^{-2\iota H_L \Delta t} e^{-\iota H_{L-1} \Delta t} \dots e^{-\iota H_1 \Delta t}) \end{aligned}$$

Because $[H_L, H_L] = 0$, thus we can write $e^{-\iota H_L \Delta t} e^{-\iota H_L \Delta t} = e^{-2\iota H_L \Delta t}$. Now applying the formula for terms, $e^{-2\iota H_{L-1} \Delta t / 2} e^{-2\iota H_L \Delta t} e^{-2\iota H_{L-1} \Delta t / 2} = e^{2\iota(H_{L-1} + H_L)\Delta t} + \mathcal{O}(\Delta t^3)$, thus using this formula recursively, say for a term

$$e^{-2\iota H_j \Delta t / 2} e^{-2\iota \sum_{k=j+1}^L H_k \Delta t} e^{-2\iota H_j \Delta t / 2} = e^{-2\iota \sum_{k=j}^L H_k \Delta t} + \mathcal{O}(\Delta t^3)$$

thus, we get,

$$\begin{aligned} U_{\Delta t} &= e^{-2\iota H_1 \Delta t / 2} e^{-2\iota \sum_{k=2}^L H_k \Delta t} e^{-2\iota H_1 \Delta t / 2} + \mathcal{O}(\Delta t^3) \\ &= e^{-2\iota \sum_{k=1}^L H_k \Delta t} + \mathcal{O}(\Delta t^3) \\ &= e^{-2\iota H \Delta t} + \mathcal{O}(\Delta t^3) \end{aligned}$$

Hence proved.

(b) Using the result from Box 4.1 as stated below

$$E(U_m U_{m-1} \dots U_1, V_m V_{m-1} \dots V_1) \leq \sum_{j=1}^m E(U_j, V_j)$$

Thus,

$$E(U_{\Delta t}^m, e^{-2m\iota H \Delta t}) = E(U_{\Delta t} U_{\Delta t} \dots U_{\Delta t}, e^{-2\iota H \Delta t} e^{-2\iota H \Delta t} \dots e^{-2\iota H \Delta t})$$

can be simplified as follows

$$\begin{aligned} E(U_{\Delta t} U_{\Delta t} \dots U_{\Delta t}, e^{-2\iota H \Delta t} e^{-2\iota H \Delta t} \dots e^{-2\iota H \Delta t}) &\leq \sum_{j=1}^m E(U_{\Delta t}, e^{-2\iota H \Delta t}) \\ &\leq \sum_{j=1}^m \max_{|\psi\rangle} \|(U_{\Delta t} - e^{-2\iota H \Delta t}) |\psi\rangle\| \\ &\leq \sum_{j=1}^m \max_{|\psi\rangle} \|\mathcal{O}(\Delta t^3) |\psi\rangle\| \\ &\leq m\alpha\Delta t^3 \end{aligned}$$

Here we have used the fact as derived in part (a) that $E(U_{\Delta t}, e^{-2\imath H \Delta t}) = \alpha \Delta t^3$. For some constant α because of the term $\mathcal{O}(\Delta t^3)$, we thus get the following result

$$E(U_{\Delta t}^m, e^{-2\imath m H \Delta t}) \leq m \alpha \Delta t^3$$

Hence, proved.

We can thus apply this formula to the k-local Hamiltonians even when they do not commute with each other.

$$e^{-\imath \sum_{l=1}^L \hat{H}_l t} = \lim_{n \rightarrow \infty} \left(\prod_{e=1}^L e^{-\imath \hat{H}_l \frac{t}{n}} \right)^n$$

Note that this formula involves infinity so we need to truncate the series when implementing this formula on a quantum computer. It is this truncation that introduces error in the simulation

$$e^{-\imath \sum_{l=1}^L \hat{H}_l t} \approx \left(\prod_{l=1}^L e^{-\imath \hat{H}_l \frac{t}{n}} \right)^n$$

This truncation and application of Suzuki-trotter formula to Hamiltonian simulation problem is known as Trotterization and its widely used to simulate non-commuting Hamiltonians on Quantum Computers. When the local terms of the hamiltonians commute the identity is exact. Therefore to perform Hamiltonian simulation to time t , the error is

$$\|e^{-\imath t H} - \left(e^{-\imath \frac{t}{L} H_1} e^{-\imath \frac{t}{L} H_2} \right)^L \| = \mathcal{O}(\Delta t^2 L) = \mathcal{O}\left(\frac{t^2}{L}\right)$$

Note that the error term is actually $\mathcal{O}(\|H_1\| \|H_2\| t^2 / n)$. So to reach precision ϵ in the operator norm, we need

$$L = \mathcal{O}(t^2 \epsilon^{-1})$$

Thus, the number of trotter steps is L , and is chosen such that the simulation error reaches the desired value. This can be improved to the second order Trotter method (also called the symmetric trotter splitting or Strang splitting)

$$\|e^{-\imath \Delta t H} - e^{-\imath \Delta t/2 H_2} e^{-\imath \Delta t H_1} e^{-\imath \Delta t/2 H_2} \| = \mathcal{O}(\Delta t^3)$$

Following a similar analysis to the first order method, we find that to reach precision ϵ we need

$$L = \mathcal{O}(t^{3/2} \epsilon^{-1/2})$$

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 ϵ , 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 $p \rightarrow \infty$. Below given is the algorithm for simulating the Quantum Systems, along with an explicit example of simulation the one-dimensional non-relativistic Schrodinger equation.

Algorithm 10 Quantum Simulation

- 1: **Input:** A Hamiltonain $H = \sum_k H_k$ acting on an N -dimensional system, where each H_k acts on a small subsystem of size independent of N , (2) An initial state $|\psi_0\rangle$, of the system at $t = 0$. (3) A positive, non-zero accuracy δ , and (3) A time t_f at which the evolved state is desired.
 - 2: **Output:** A state $|\tilde{\psi}(t_f)\rangle$ such that $|\langle \tilde{\psi}(t_f) | e^{-\iota H t_f} |\psi_0\rangle|^2 \geq 1 - \delta$
 - 3: **Runtime:** $\mathcal{O}(\text{poly}(1/\delta))$
 - 4: **Procedure:** Choose a representation such that the state $|\tilde{\psi}\rangle$ of $n = \text{poly}(\log N)$ qubits approximates the system and the operators $e^{-\iota H_k \Delta t}$ have efficient quantum circuit approximation. Select an approximation method and Δt such that the expected error is acceptable (and $j\Delta t = t_f$ for an integer j), construct the corresponding quantum circuit $U_{\Delta t}$ for the iterative step and do:
 1. **Initialize State:** $|\tilde{\psi}_0\rangle \leftarrow |\psi_0\rangle ; j = 0$
 2. **Iterative Update:** $\rightarrow |\tilde{\psi}_{j+1}\rangle = U_{\Delta t} |\tilde{\psi}_j\rangle$
 3. **Loop:** $\rightarrow j = j + 1$; goto 2 until $j\Delta t \geq t_f$
 4. **Final Result:** $\rightarrow |\tilde{\psi}(t_f)\rangle = |\tilde{\psi}_j\rangle$
-

Example 20.5.2. Say Hamiltonian \hat{H} we want to simulate is a sum of two Pauli matrices $\hat{H} = \hat{X} + \hat{Z}$ which acts on a single qubit. Note that it is small enough to do an exact simulation and compare it with a simulation using trotterization for different no. of Trotter steps. The matrix is simultaneous rotation around X and Z axis of the Bloch sphere. Recall that because the Pauli X and Pauli Z do not commute i.e. $[X, Z] = 2\iota Y$, thus $e^{-\iota H t} = e^{-\iota(X+Z)t} \neq e^{-\iota X t} e^{-\iota Z t}$. Thus, we use trotterization. Say we use first-order trotterization as follows:

$$e^{-\iota H t} = \left(e^{-\iota X \frac{t}{n}} e^{-\iota Z \frac{t}{n}} \right)^n$$

We, thus evolve the Hamiltonian repeatedly by switching between evolving X and Z each for a small period of time. Thus, the next step is to represent each product in Trotter's formula as a quantum circuit. In this case it is simple because evolution can be represented exactly using Rotations around X and Z axis. Recall,

$$e^{-\iota \frac{\theta}{2} X} = R_x(\theta) = \begin{pmatrix} \cos(\theta/2) & -\iota \sin(\theta/2) \\ -\iota \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$$

and

$$e^{-\iota \frac{\theta}{2} Z} = R_Z(\theta) = \begin{pmatrix} e^{-\iota \frac{\theta}{2}} & 0 \\ 0 & e^{\iota \frac{\theta}{2}} \end{pmatrix}$$

Rotation angle then encodes the evolution. Putting $\theta = \frac{2t}{n}$, we get,

$$R_x\left(\frac{2t}{n}\right) = e^{-\iota \frac{t}{n} X} = \begin{pmatrix} \cos(\frac{t}{n}) & -\iota \sin(\frac{t}{n}) \\ -\iota \sin(\frac{t}{n}) & \cos(\frac{t}{n}) \end{pmatrix}; \quad R_Z\left(\frac{2t}{n}\right) = e^{-\iota \frac{t}{n} Z} = \begin{pmatrix} e^{-\iota \frac{t}{n}} & 0 \\ 0 & e^{\iota \frac{t}{n}} \end{pmatrix}$$

The more the terms are commuting between each other, the more precise the simulation is. The 1st order formula is good for short time intervals. Note that we did not require any ancilla bits.

Example 20.5.3. (Baker-Campbell-Hausdorff formula) Prove that

$$e^{(A+B)\Delta t} = e^{A\Delta t} e^{B\Delta t} e^{-\frac{1}{2}[A,B]\Delta t^2} + \mathcal{O}(\Delta t^3)$$

and also prove that $e^{\iota(A+B)\Delta t} = e^{\iota A\Delta t} e^{\iota B\Delta t} + \mathcal{O}(\Delta t^2)$ and $e^{\iota(A+B)\Delta t} = e^{\iota A\Delta t/2} e^{\iota B\Delta t} e^{\iota A\Delta t/2} + \mathcal{O}(\Delta t^3)$.

We will begin with proving $e^{\iota(A+B)\Delta t} = e^{\iota A\Delta t} e^{\iota B\Delta t} + \mathcal{O}(\Delta t^2)$. Recall, from the trotter formula proof that

$$e^{\iota At/n} e^{\iota Bt/n} = I + \frac{1}{n} \iota(A + B)t + \mathcal{O}\left(\frac{1}{n^2}\right)$$

Putting $t = n\Delta t \implies n = \frac{t}{\Delta t}$, we get,

$$\begin{aligned} e^{\iota A\Delta t} e^{\iota B\Delta t} &= I + \iota(A + B)\Delta t + \mathcal{O}(\Delta t^2) \\ e^{\iota A\Delta t} e^{\iota B\Delta t} - \mathcal{O}(\Delta t^2) &= e^{\iota(A+B)\Delta t} \\ \implies e^{\iota(A+B)\Delta t} &= e^{\iota A\Delta t} e^{\iota B\Delta t} + \mathcal{O}(\Delta t^2) \end{aligned}$$

Now proving $e^{\iota(A+B)\Delta t} = e^{\iota A \Delta t / 2} e^{\iota B \Delta t} e^{\iota A \Delta t / 2} + \mathcal{O}(\Delta t^3)$

$$\begin{aligned} e^{\iota A \Delta t / 2} e^{\iota B \Delta t} e^{\iota A \Delta t / 2} &= \left(I + \iota A \Delta t / 2 + \frac{(\iota A \Delta t / 2)^2}{2!} \right) \left(I + \iota B \Delta t + \frac{(\iota B \Delta t)^2}{2!} \right) \\ &\quad \left(I + \iota A \Delta t / 2 + \frac{(\iota A \Delta t / 2)^2}{2!} \right) \\ &= I + \iota(A+B)\Delta t + \frac{(\iota(A+B)\Delta t)^2}{2!} + \mathcal{O}(\Delta t^3) \\ \implies e^{\iota(A+B)\Delta t} &= e^{\iota A \Delta t / 2} e^{\iota B \Delta t} e^{\iota A \Delta t / 2} + \mathcal{O}(\Delta t^3) \end{aligned}$$

Now proving the baker-Campbell-Hausdorff formula,

$$\begin{aligned} e^{A\Delta t} e^{B\Delta t} e^{-\frac{AB-BA}{2}\Delta t^2} &= \left(I + A \Delta t + \frac{(A \Delta t)^2}{2} \right) \left(I + B \Delta t + \frac{(B \Delta t)^2}{2} \right) \left(I - \frac{AB-BA}{2} \Delta t^2 \right) \\ &= I + A \Delta t + B \Delta t + \frac{(A \Delta t)^2}{2} + \frac{(B \Delta t)^2}{2} + AB \Delta t^2 - \frac{AB \Delta t^2}{2} \\ &\quad + \frac{BA \Delta t^2}{2} + \mathcal{O}(\Delta t^3) \\ &= I + (A+B)\Delta t + \frac{A^2 + B^2 + AB + BA}{2} \Delta t^2 + \mathcal{O}(\Delta t^3) \\ &= I + (A+B)\Delta t + (A+B)^2 \frac{\Delta t^2}{2} + \mathcal{O}(\Delta t^3) \\ \implies e^{(A+B)\Delta t} &= e^{A\Delta t} e^{B\Delta t} e^{-\frac{AB-BA}{2}\Delta t^2} + \mathcal{O}(\Delta t^3) \end{aligned}$$

Example 20.5.4. (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

$$e^{-\iota t H_1} := e^{\iota t} \sum_{i=1}^{n-1} Z_i Z_{i+1} = \prod_{i=1}^{n-1} e^{\iota t Z_i Z_{i+1}}$$

Each $e^{\iota t Z_i Z_{i+1}}$ is a rotation involving only the qubits i, j , and the splitting has no error. Similarly

$$e^{-\iota t H_2} := e^{d \sum_i i X_i} = \prod_i e^{\iota t g X_i}$$

and each $e^{\iota t g X_i}$ can be implemented independently without error.

Example 20.5.5. (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, $e^{\iota H_1 t}$ can be efficiently performed by diagonalizing H_1 in the Fourier space, and $e^{\iota H_2 t}$ can be efficiently performed since $V(\mathbf{r})$ is diagonal in the real space.

20.6 Evolution of Hamiltonian formed by Pauli Operators: Quantum circuits and exact simulation

Evolution generated by some non-local Hamiltonians can be simulated exactly. Such a possibility occurs when a hamiltonian consists of Tensor product of Pauli operators. Examples of such Hamiltonians are $\hat{H} = \hat{X}$, $\hat{H} = \hat{Z}_1 \otimes \hat{Z}_2$, $\hat{H} = \hat{X}_1 \otimes \hat{Y}_2 \otimes \hat{Z}_3$. In total any Hamiltonian of the form:

$$\hat{H} = \otimes_{k=1}^n \sigma_{\alpha k}^k$$

We will now see how to construct a quantum circuit a Quantum circuit to simulate a Hamiltonian consists of an arbitrary tensor product of Pauli operators.

20.6.1 One-qubit Hamiltonians

In other words, when the Hamiltonians is formed by one of the Pauli operators ($\hat{H} = \{X, Y, Z, I\}$). In this case, the evolution operator is just the corresponding rotation Operators.

$$U = e^{-i\hat{H}t}$$

$$\begin{aligned} R_x(\theta) &= e^{-i\frac{\theta}{2}\hat{X}} = \cos \frac{\theta}{2} \hat{I} - i \sin \frac{\theta}{2} \hat{Y} = \begin{pmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix} \\ R_Y(\theta) &= e^{-i\frac{\theta}{2}\hat{Y}} = \cos \frac{\theta}{2} \hat{I} - i \sin \frac{\theta}{2} \hat{Y} = \begin{pmatrix} \cos \frac{\theta}{2} & \sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix} \\ R_Z(\theta) &= e^{-i\frac{\theta}{2}\hat{Z}} = \cos \frac{\theta}{2} \hat{I} - i \sin \frac{\theta}{2} \hat{Z} = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix} \end{aligned}$$

20.6.2 Two-qubit Hamiltonians

Now consider the situation when the Hamiltonian consists of Pauli operators applied to two qubits. Say $\hat{H} = \hat{Z}_1 \otimes \hat{Z}_2$. Here, the subscript denotes the qubit it acts on. The Hamiltonian is as follows:

$$\hat{Z}_1 \otimes \hat{Z}_2 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

20.6. EVOLUTION OF HAMILTONIAN FORMED BY PAULI OPERATORS: QUANTUM CIRCUIT

Note that the square of this matrix is an Identity matrix. Therefore, to calculate the evolution operator of a given Hamiltonian we can use the formula for the decomposition of the matrix exponential.

$$U = e^{-\iota(\hat{Z}_1 \otimes \hat{Z}_2)t} = \cos t \hat{I}_1 \otimes \hat{I}_2 - \iota \sin t \hat{Z}_1 \otimes \hat{Z}_2$$

$$= \begin{pmatrix} \cos t - \iota \sin t & 0 & 0 & 0 \\ 0 & \cos t + \iota \sin t & 0 & 0 \\ 0 & 0 & \cos t + \iota \sin t & 0 \\ 0 & 0 & 0 & \cos t - \iota \sin t \end{pmatrix}$$

using the exponential notation of the colex number, we get,

$$= \begin{pmatrix} e^{-\iota t} & 0 & 0 & 0 \\ 0 & e^{\iota t} & 0 & 0 \\ 0 & 0 & e^{\iota t} & 0 \\ 0 & 0 & 0 & e^{-\iota t} \end{pmatrix}$$

$$\hat{R}_{ZZ}(\theta) = e^{-\iota \frac{\theta}{2} \hat{Z}_1 \otimes \hat{Z}_2} = \begin{pmatrix} e^{-\iota \frac{\theta}{2}} & 0 & 0 & 0 \\ 0 & e^{\iota \frac{\theta}{2}} & 0 & 0 \\ 0 & 0 & e^{\iota \frac{\theta}{2}} & 0 \\ 0 & 0 & 0 & e^{-\iota \frac{\theta}{2}} \end{pmatrix}$$

Thus, we have an operator called R_{ZZ} in the literature and appears in the QAOA algorithm. Our goal is to construct a quantum circuit that implements this operator. To do this, we first look at how the R_{ZZ} operator acts on a state with normalized coefficients. $|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$. For $\hat{H} = \hat{Z}_1 \otimes \hat{Z}_2$

$$R_{ZZ} |\psi\rangle = \begin{pmatrix} e^{-\iota \frac{\theta}{2}} & 0 & 0 & 0 \\ 0 & e^{\iota \frac{\theta}{2}} & 0 & 0 \\ 0 & 0 & e^{\iota \frac{\theta}{2}} & 0 \\ 0 & 0 & 0 & e^{-\iota \frac{\theta}{2}} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} \alpha e^{-\iota \frac{\theta}{2}} \\ \beta e^{\iota \frac{\theta}{2}} \\ \gamma e^{\iota \frac{\theta}{2}} \\ \delta e^{-\iota \frac{\theta}{2}} \end{pmatrix}$$

$$= \alpha e^{-\iota \frac{\theta}{2}} |00\rangle + \beta e^{\iota \frac{\theta}{2}} |01\rangle + \gamma e^{\iota \frac{\theta}{2}} |10\rangle + \delta e^{-\iota \frac{\theta}{2}} |11\rangle$$

Now, taking the Global phase factor, we get,

$$= e^{-\iota \frac{\theta}{2}} [\alpha|00\rangle + \beta e^{\iota \theta}|01\rangle + \gamma e^{\iota \theta}|10\rangle + \delta|11\rangle]$$

Thus, we can see that the action of the R_{ZZ} i.e. the Hamiltonian $\hat{H} = \hat{Z}_1 \otimes \hat{Z}_2$ is a simultaneous rotation of the two qubits around Z axis. From the result we just

received one can see that the phase shift applied to the system is $e^{-\iota \frac{\theta}{2}}$ if the parity of qubits in the computational basis is even i.e. both the qubits are either zero or one. Other wise the phase shift should be $e^{\iota \frac{\theta}{2}}$. Thus, for the simple simulation of the ZZ Hamiltonain, we need to do the parity check and then apply a phase shift conditioned on the parity and then compute.

20.7 How to simulate any Hamiltonian

The following are the steps to simulate any Hamiltonian. Sometimes this approach may not be optimal but it works for most of the models of interest.

Assume that the Hamiltonian simulation will be performed on a universal quantum computer with a gate based model of computation.

1. Define a model Hamiltonian of the system which we want to simulate.

$$U = e^{-\iota \hat{H}t}$$

2. Write the target Hamiltonian \hat{H} in terms of Pauli matrices. the mathematical properties of qubits are those of spin-1/2 systems. Therefore in order to be simulated on a qubit based architecture any target Hamiltonian for example, if you want to simulate spin 1 system has to be mapped into an equivalent Hamiltonian of interacting spin half operators. Efficient mappings are known for large variety of cases ranging from spin more than half fermionic and bosonic systems. Since any hamiltonian is a matrix it can be decomposed into linear combination of products of Pauli operators.

$$\hat{H} = \begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{bmatrix}$$

We known that,

$$\hat{H} = \sum_{l=1}^L \hat{H}_l$$

Thus, we can write

$$\hat{H}_l = c_l \hat{A}_l$$

Here, \hat{A}_l is a tensor product of Pauli operators as

$$\hat{A}_l = \otimes_{k=1}^n \sigma_{\alpha k}^{kl}$$

and $c_l = \frac{\text{Tr}(\hat{A}_l \hat{H})}{2^n}$ are the expansion coefficients for all the possible combinations from the Pauli operators. In this way you can find the decomposition of the matrix into a linear combination of the power operators using the commands in Qiskit. Remember the Quantum Simulation will be efficient whenever the Hamiltonian is the sum of local terms. This is not a limitation in many practical cases because most of the physical processes are inherently local in nature. Sometime local hamiltonians get mapped into non-local ones for example when Jordan Wegner transformation is applied.

3. Trotterisation: If $[\hat{H}_i, \hat{H}_j] = \hat{H}_i \hat{H}_j - \hat{H}_j \hat{H}_i = 0$ then

$$e^{-\iota \hat{H}t} = e^{-\iota \hat{H}_1 t} \dots e^{-\iota \hat{H}_L t}$$

else

$$\hat{U}(t) = e^{-\iota \sum_{l=1}^L \hat{H}_l t} = \left(\prod_{l=1}^L e^{-\iota \hat{H}_l \frac{t}{n}} \right)^n + O\left(\frac{t^2}{n}\right)$$

4. Translate each $e^{-\iota \hat{H}_l t}$ into a quantum circuit. Each local term in this case can be simulated exactly.
5. Initial State and Measurements: It is optional. Sometimes you may need to add initial state preparation at the beginning of the circuit and the appropriate set of measurements at the end to recover the expectation values of the observable quantities. We can get the expectation value of the observable using hadamard test.

20.8 Example

The procedure so far described for quantum simulations has concentrated on simulating Hamiltonians which are sums of local interactions. However, this is not a fundamental requirement. As the following example illustrates, efficient quantum simulations are possible even for Hamiltonians which act non-trivially on all or nearly all parts of a large system.

Suppose we have the Hamiltonian

$$H = Z_1 \otimes Z_2 \otimes \dots \otimes Z_n$$

which acts on an n qubit system. Despite this being an interaction involving all of the system, indeed, it can be simulated efficiently. What we desire is a simple quantum

circuit which implements $e^{-\imath H \Delta t}$, for arbitrary values of Δt . A circuit doing precisely this, for $n = 3$ is as shown in the figure 20.1. The main insight is that although the Hamiltonians involved all the qubits in the system, it does so in a classical manner: the phase shift applied to the system is $e^{-\imath \Delta t}$ if the parity of the n qubits in the computational basis is even; otherwise, the phase shift should be $e^{\imath \Delta t}$. Thus, simple simulation of H is possible by first classically computing the parity (storing the result in an ancilla qubit), then applying the appropriate phase shift conditioned on the parity, then uncomputing the parity (to erase the ancilla). This strategy clearly works not only for $n = 3$, but also for arbitrary values of n . Furthermore, extending

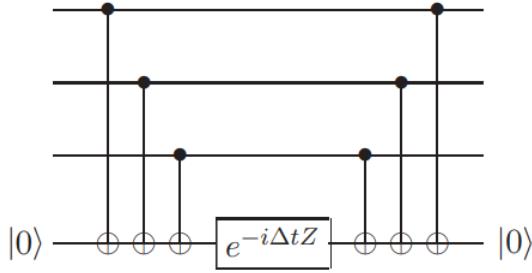


Figure 20.1: Quantum circuit for simulating the Hamiltonian $H = Z_1 \otimes Z_2 \otimes Z_3$ for time Δt

the same procedure allows us to simulate more complicated extended Hamiltonians. Specifically, we can efficiently simulate any Hamiltonian of the form

$$H = \otimes_{k=1}^n \sigma_{c(k)}^k$$

where $\sigma_{c(k)}^k$ is a Pauli matrix (or the identity) acting on the k th qubit, with $c(k) \in \{0, 1, 2, 3\}$ specifying one of $\{I, X, Y, Z\}$. The qubits upon which the identity operation is performed can be disregarded, and X , or Y terms can be transformed by single qubit gates to Z operations. This leaves us with a Hamiltonian of the form, which is simulated as described above.

The circuit shown in figure 20.1 simulates $e^{-\imath H \Delta t} = e^{-\imath \Delta t(Z_1 \otimes Z_2 \otimes Z_3)}$. All this can be done because we use the property that $e^{\imath Ax} = \cos xI - \imath \sin xA$ when $A^2 = I$. Note that the $(\sigma_1 \otimes \sigma_2 \otimes \dots \otimes \sigma_n)^2 = I$ where σ_i are any one of the Pauli matrices acting on their respective qubits. This is because $(\otimes_{i=1}^n \sigma_i)(\otimes_{i=1}^n \sigma_i) = (\otimes_{i=1}^n \sigma_i^2) = I$.

This can be shown as follows:

$$\begin{aligned}
e^{-\iota \Delta t Z_1 \otimes Z_2 \otimes Z_3} |abc\rangle &= (\cos \Delta t I - \iota \sin \Delta t (Z_1 \otimes Z_2 \otimes Z_3)) |abc\rangle \\
&= (\cos \Delta t |abc\rangle - \iota \sin \Delta t (Z_1 \otimes Z_2 \otimes Z_3) |abc\rangle \\
&= (\cos \Delta t |abc\rangle - \iota \sin \Delta t (-1)^{a+b+c} |abc\rangle \\
&= \cos \Delta t |abc\rangle - (-1)^{a+b+c} \iota \sin \Delta t |abc\rangle \\
&= (\cos \Delta t - (-1)^{a+b+c} \iota \sin \Delta t) |abc\rangle \\
&= \begin{cases} e^{-\iota \Delta t} |abc\rangle & \text{if the parity is even for } n \text{ qubits} \\ e^{\iota \Delta t} |abc\rangle & \text{if the parity is odd for } n \text{ qubits} \end{cases}
\end{aligned}$$

which is what is implemented by the circuit shown in figure 20.1, where the result will be $e^{-\iota \Delta t}$ if the parity is even for n qubits and $e^{\iota \Delta t}$ if the parity is odd for n qubits.

Example 20.8.1. Construct a quantum circuit to simulate the Hamiltonian

$$H = X_1 \otimes Y_2 \otimes Z_3$$

performing the unitary transformation $e^{-\iota \Delta t H}$ for any Δt .

Recall the matrix identities,

$$X = HZH; \quad Y = R_X \left(-\frac{\pi}{2} \right) Z R_X \left(\frac{\pi}{2} \right)$$

Thus, using this single qubit gates we can write the Hamiltonian as follows:

$$H = (H \otimes R_X \left(-\frac{\pi}{2} \right) \otimes I)(Z_1 \otimes Z_2 \otimes Z_3)(H \otimes R_X \left(\frac{\pi}{2} \right) \otimes I)$$

Now, we are required to simulate $U = e^{-\iota(X_1 \otimes Y_2 \otimes Z_3)\Delta t}$. We can thus, simplify this as follows:

$$\begin{aligned}
U &= e^{-\iota(X_1 \otimes Y_2 \otimes Z_3)\Delta t} \\
&= \cos \Delta t I_1 \otimes I_2 \otimes I_3 - \iota \sin \Delta t X_1 \otimes Y_2 \otimes Z_3 \\
&= (H_1 \otimes R_X \left(-\frac{\pi}{2} \right) \otimes I)(\cos \Delta t I_1 \otimes I_2 \otimes I_3 - \iota \sin \Delta t Z_1 \otimes Z_2 \otimes Z_3)(H_1 \otimes R_X \left(\frac{\pi}{2} \right) \otimes I_3) \\
&= (H_1 \otimes R_X \left(-\frac{\pi}{2} \right) \otimes I)e^{-\iota Z_1 \otimes Z_2 \otimes Z_3 \Delta t}(H_1 \otimes R_X \left(\frac{\pi}{2} \right) \otimes I_3)
\end{aligned}$$

Using this procedure allows us to simulate a wide class of Hamiltonians containing terms which are not local. In particular, it is possible to simulate a wide class of hamiltonians containing terms which are not local. In particular, it is possible to

simulate a Hamiltonian of the form $H = \sum_{k=1}^L H_k$ where the only restriction is that the individual H_k have a tensor product structure, and that L is a polynomial in the total number of particles n . More generally, all that is required is that there be an efficient circuit to simulate each H_k separately. As an example, the Hamiltonian $H = \sum_{k=1}^n X_k + Z^{\otimes n}$ can be easily simulated using the above techniques. Such Hamiltonians typically do not arise in Nature. However, they provide a new and possibly valuable vista on the world of quantum algorithms.

20.9 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 $\tilde{U}(t) = e^{-\iota tH_1}e^{-\iota tH_2}$ satisfies the equation

$$\begin{aligned}\iota\partial_t\tilde{U}(t) &= e^{-\iota tH_1}e^{-\iota tH_2} + e^{-\iota tH_1}H_2e^{-\iota tH_2} \\ &= (H_1 + H_2)e^{-\iota tH_1}e^{-\iota tH_2} + e^{-\iota tH_1}H_2e^{-\iota tH_2} - H_2e^{-\iota tH_1}e^{-\iota tH_2} \\ &= H\tilde{U}(t) + [e^{-\iota tH_1}, H_2]e^{-\iota tH_2}\end{aligned}$$

with initial condition $\tilde{U}(0) = I$. By Duhamel's principle, and let $U(t) = e^{-\iota tH}$, we have

$$\tilde{U}(t) = U(t) - \iota \int_0^t e^{-\iota H(t-s)}[e^{-\iota sH_1}, H_2]e^{-\iota sH_2}ds$$

So we have

$$\|\tilde{U}(t) - U(t)\| \leq \int_0^t \|e^{-\iota sH_1}, H_2\| ds$$

Now consider $G(t) = [e^{-\iota tH_1}, H_2]e^{\iota tH_1} = e^{-\iota tH_1}H_2e^{\iota tH_1} - H_2$, which satisfies $G(0) = 0$ and

$$\iota\partial_t G(t) = e^{-\iota tH_1}[H_1, H_2]e^{+\iota tH_1}$$

Hence,

$$\|[e^{-\iota tH_1}, H_2]\| = \|G(t)\| \leq t\|[H_1, H_2]\|$$

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 ϵ , 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

$$\|[H_1, H_2]\psi\| \leq \|V''\| + \|V'\| \|\psi'\| = \mathcal{O}(N)$$

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

$$\|e^{-\iota|\Delta t| H} \psi(0) - e^{i\omega \Delta t H_1} e^{-\iota\omega \Delta t H_2} \psi(0)\| \leq \frac{\Delta t^2}{2} \max_{0 \leq s \leq \Delta t} \|[H_1, H_2]\psi(s)\|$$

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 20.9.1. 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 20.9.2. 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 Δt 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 21

The HHL Algorithm

21.1 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.

21.2 The Linear-System Problem (LSP)

The classical linear-system problem is defined as follows:

Definition 21.2.1. 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, κ is the condition number of the system and ϵ 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 21.2.2. 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 ϵ 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 κ is some finite value) The ratio between its largest and smallest singular value is at most some κ . For simplicity, assume the smallest singular value is $\geq 1/\kappa$ while the largest is ≤ 1 . In other words, all eigenvalues of A lie in the interval $[-1, -1/\kappa] \cup [1/\kappa, 1]$. The smaller the condition number κ is, the better it will be for the algorithm. Let's assume our algorithm knows κ , or at least knows a reasonable upper bound on κ .

21.3 The HHL Algorithm

21.3.1 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 Λ 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 λ_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 $A|x\rangle = |b\rangle$.

21.3.2 Algorithm

Consider the following circuit as shown in the figure 21.1. 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.

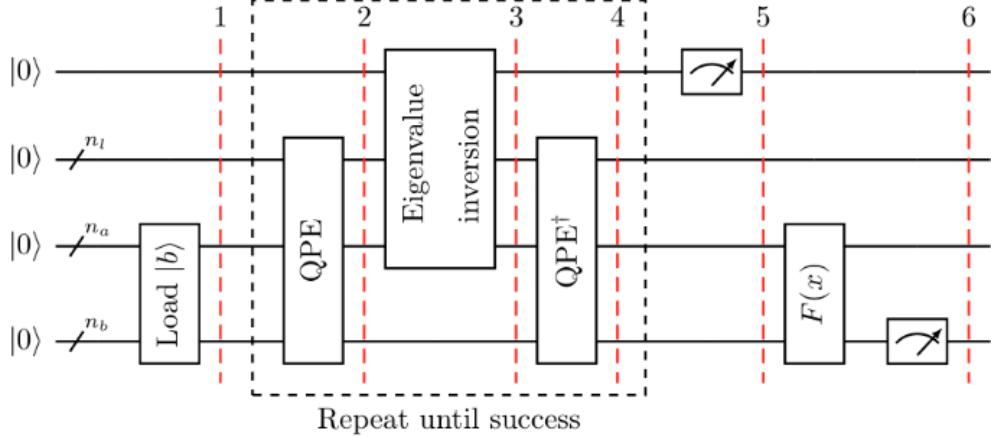


Figure 21.1: HHL Circuit

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 θ 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 θ_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 21.2. Note that the



Figure 21.2: QPE

input to the QPE is not an eigen state but as shown in the figure 21.2, 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 θ_j . Thus, the output of the QPE will be a superposition of the n_l bit binary approximations of the actual phases θ_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 21.3.1. (*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_θ 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

$$\begin{aligned} U_\theta &= \sum_{j=0}^{N-1} e^{-i\pi(0.j_{d-1} \dots j_0)\sigma_Y} \otimes |j\rangle \langle j| \\ &= \sum_{\theta_{d-1}, \dots, \theta_0} e^{-i\pi(0.\theta_{d-1} \dots \theta_0)\sigma_Y} \otimes |\theta_{d-1} \dots \theta_0\rangle \langle \theta_{d-1} \dots \theta_0| \end{aligned}$$

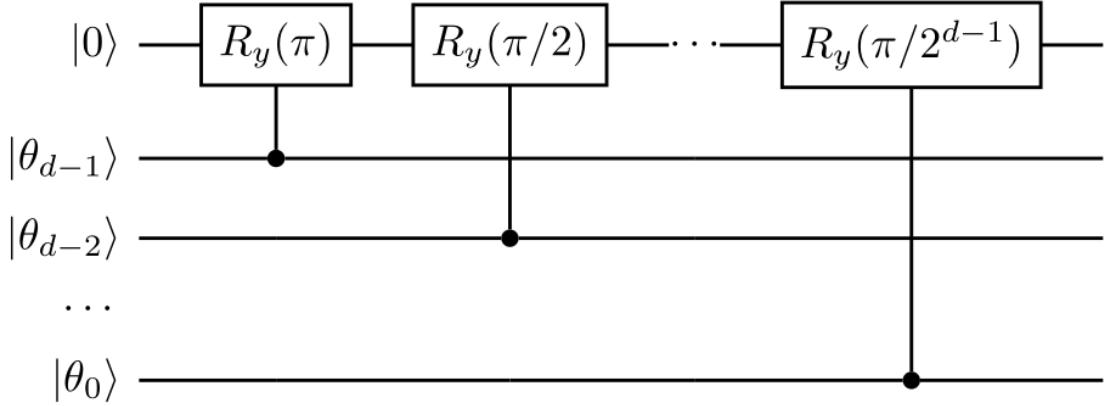


Figure 21.3: Controlled Rotation

$$\begin{aligned}
 U_\theta &= \sum_{\theta_{d-1}, \dots, \theta_0} e^{-i\pi\frac{\theta_{d-1}}{2}\sigma_Y} e^{-i\pi\frac{\theta_{d-2}}{4}\sigma_Y} \dots e^{-i\pi\frac{\theta_0}{2^d}\sigma_Y} \otimes |\theta_{d-1} \dots \theta_0\rangle \langle \theta_{d-1} \dots \theta_0| \\
 &= \sum_{\theta_{d-1}, \dots, \theta_0} R_Y(\pi\theta_{d-1}) R_Y(\pi\frac{\theta_{d-2}}{2}) \dots R_Y(\pi\frac{\theta_0}{2^{d-1}}) \otimes |\theta_{d-1} \dots \theta_0\rangle \langle \theta_{d-1} \dots \theta_0| \\
 &= \sum_{\theta_{d-1}, \dots, \theta_0} R_Y(\pi\theta_{d-1}) R_Y(\frac{\pi}{2}\theta_{d-2}) \dots R_Y(\frac{\pi}{2^{d-1}}\theta_0) \otimes |\theta_{d-1} \dots \theta_0\rangle \langle \theta_{d-1} \dots \theta_0|
 \end{aligned}$$

Thus, applying U_θ to $|0\rangle|\theta\rangle$ gives the desired result since corresponding to a particular value of θ 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 21.3. \square

Thus, as shown in the circuit in figure 21.3 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 θ_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.

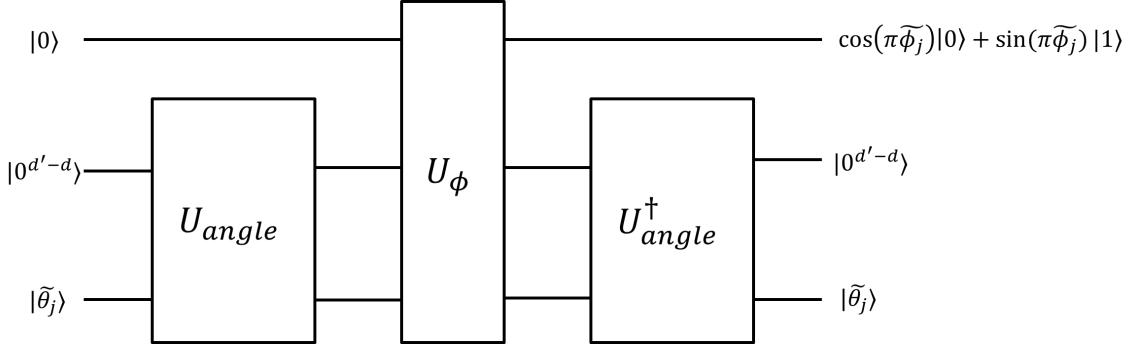


Figure 21.4: Entire Controlled Rotation

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 λ'_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 21.4. 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

$$p(1) = \left\| \sum_j \frac{C b_j}{\tilde{\theta}_j'} \right\|^2 = \|\tilde{x}\|^2$$

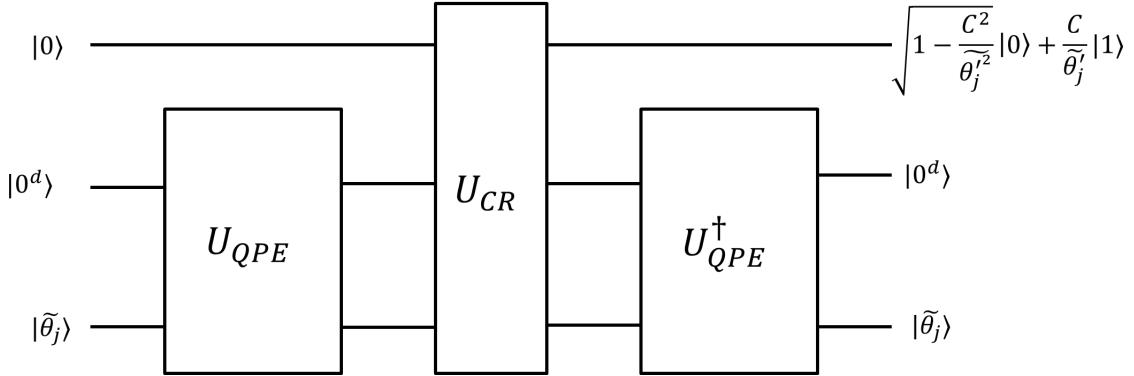


Figure 21.5: HHL Circuit

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 21.5.

21.3.3 Example

Example 21.3.1. (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,

$$\begin{aligned} &= \frac{1}{\sqrt{2}} \left(\frac{\sqrt{3}}{2} |0\rangle + \frac{1}{2} |1\rangle \right) \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) + \frac{1}{\sqrt{2}} \left(\frac{\sqrt{15}}{4} |0\rangle + \frac{1}{4} |1\rangle \right) \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ &= \frac{\sqrt{3}}{4} |00\rangle - \frac{\sqrt{3}}{4} |01\rangle + \frac{1}{4} |10\rangle - \frac{1}{4} |11\rangle + \frac{\sqrt{15}}{8} |00\rangle + \frac{\sqrt{15}}{8} |01\rangle + \frac{1}{8} |10\rangle \\ &\quad + \frac{1}{8} |11\rangle \end{aligned}$$

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 .

21.4 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 λ_0). So assuming that the exact knowledge of λ_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 λ_{max} is the eigen value with the largest magnitude (irrespective of sign)) which is also positive definite (has all the eigen values ≥ 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 ϵ , we need to estimate the eigen values to multiplicative accuracy ϵ instead of the standard additive accuracy. **An ϵ -additive precision is an algorithm that for a true solution x returns $\tilde{x} \in [x - \epsilon, x + \epsilon]$. For a ϵ -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 ψ_{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 λ_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 κ , 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 ϵ . 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 ϵ of the observable, the number of samples needed is $\mathcal{O}(\epsilon^{-2})$. On the other hand,

in order to reach precision ϵ , the solution vector $|\tilde{x}\rangle$ must be solved to precision ϵ . 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 κ 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.

21.5 Improvements in HHL algorithm

The complexity of the above basic HHL algorithm can be improved further. Gilyen et al used the singular value transformation technique to implement A^{-1} , improving on an LCU construction due to Childs et al. We would like to apply the function $f(x) = 1/x$ to a block-encoding of A in order to get a block-encoding of A^{-1} (up to normalization) that we can then apply to $|b\rangle|0\rangle$.

The function $f(x) = 1/x$ is not itself a polynomial, so we need to approximate it by a low-degree polynomial to be able to apply. Childs et al started from the following polynomial of degree $D = 2b - 1$ for $b = \mathcal{O}(\kappa^2 \log(\kappa/\epsilon))$:

$$\frac{1 - (1 - x^2)^b}{x}$$

This is indeed a polynomial because all terms in the numerator have degree ≥ 1 , so we can divide out the x of the denominator. Since $(1 - x^2)^b$ is close to 0 (unless $|x|$ is small), this polynomial is indeed close to $1/x$ (unless $|x|$ is small, but we won’t care about that because we’ll apply this polynomial to a matrix whose eigenvalues aren’t close to 0). More precisely, this polynomial is $\epsilon/2$ close to $1/x$ whenever x lies in the range $E_\kappa = [-1, -1/\kappa] \cup [1/\kappa, 1]$. Its range on this domain is $[-\kappa, -1] \cup [1, \kappa]$ (ignoring the small ϵ for simplicity). Like every degree- D polynomial, f can be written exactly as a sum of $D + 1$ Chebyshev polynomials of the first kind. These univariate polynomials are defined recursively as follows: $T_0(x) = 1$, $T_1(x) = x$ and $T_{d+1} = 2xT_d(x) - T_{d-1}(x)$. Note that T_d has degree d , and maps $[-1, 1]$ to $[-1, 1]$. The polynomials T_1, \dots, T_D are linearly independent (even orthonormal in a certain

way) and hence span the set of all univariate polynomials of degree $\leq D$. Childs et al show that the coefficients in this sum decrease quickly for larger degree, and that by dropping the resulting degree- d polynomial p ϵ -approximates $1/x$ on the interval E_k , and its largest value (in absolute values) on this domain is κ . Now define the polynomial $P = p/(4\kappa)$. This has the same degree d as p , but a range $[-1/4, 1/4]$ that fits the assumption (there's a trick to ensure that the values of P are within that range even for $x \approx 0$, which we'll skip here). We can implement a block-encoding of the s -sparse matrix A/s using $O(1)$ sparse-access queries to A and $O(n)$ other gates. Using a factor $O(s)$ more work, we can turn this into a block-encoding of A itself (alternatively, we could directly invert the matrix A/s , whose singular values are $\geq 1/(\kappa s)$). With this block-encoding of A and the polynomial $P = p/(4k)$, of degree $d = O(\kappa \log(\kappa/\epsilon))$. Note that all eigenvalues of A lie in the interval E_k , where $p(x) \approx 1/x$, hence $p(A) \approx A^{-1}$ and $P(A) \approx \frac{1}{4\kappa}A^{-1}$. Then we have the block encoding of $P(A)$, at the expense of running the block-encoding of A $O(d)$ times. Using $O(\kappa)$ rounds to amplitude amplification on top of this, we can get rid of the $1/(4\kappa)$ factor and end up with essentially the state $A^{-1}|b\rangle$ normalized. Note that we need to assume a unitary to prepare $|b\rangle$ here, having just one copy of $|b\rangle$ is not enough. We cannot use oblivious amplitude amplification because that assumes we have a block-encoding of a matrix that is proportional to a unitary (or close to that), which A^{-1} is not. This gives a quantum algorithm that solves QLSP using $O(d\kappa s) = O(\kappa^2 s \log(\kappa/\epsilon))$ queries to A , and $O(\kappa s(\kappa n \log(\kappa/\epsilon) + B))$ -2 qubit gates. Note that compared to basic HHL, dependence on $1/\epsilon$ has been improved from linear to logarithmic. The dependence on κ can also be improved from quadratic to linear, using a technique called “variable-time amplitude amplification”.

The HHL algorithm can in some cases solve the QLSP exponentially faster than classical algorithms can solve the LSP. In particular, if the sparsity s , the condition number κ , and the cost B of preparing $|b\rangle$ are all $\leq \text{polylog}(N)$, and the allowed error is $\epsilon \geq 2^{-\text{polylog}(N)}$, then this improved version of the HHL algorithm uses $\text{polylog}(N)$ queries and gates to solve (in a quantum way) an N -dimensional linear system. It can also be used for other tasks, for instance approximately solving differential equations and other applications in scientific computing.

21.6 Implementation using Qiskit

21.7 Applications

21.7.1 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 21.7.1. (*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 $v_{j,k} = \sin \frac{jk\pi}{N+1} e^{ij\theta}$. 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 $\kappa(A) \approx 4/(h^2\pi^2) \approx \kappa(A)$. There for 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 .

21.7.2 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 21.7.2. *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 21.7.3. (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

$$\|\mathcal{A}\| = \sqrt{\lambda_{\max}(\mathcal{A}^\dagger \mathcal{A})} \leq \sqrt{1 + |\zeta|^2 + 2|\zeta|} < 3$$

and

$$\|\mathcal{A}^{-1}\|^{-1} = \sqrt{\lambda_{\min}(\mathcal{A}^\dagger \mathcal{A})} \geq \frac{\Delta t(-\operatorname{Re}(a))}{2}$$

As a result, when $(-\operatorname{Re}(a)) = \Theta(1)$, 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 $\operatorname{Re}(a) < 0$, 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 $\operatorname{Re}(a) > 0$. 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 $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_N)$. We only consider the case $\operatorname{Re}(\lambda_k) < 0$ for all k .

Proposition 21.7.4. *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 22

Quantum Walk Algorithms

22.1 Classical Random Walks

Consider an undirected graph G with N vertices. Suppose at least an ϵ - fraction of the vertices are “marked”, and we would like to find a marked vertex. One way to do this is with a random walk:

1. Start at some specific vertex y of the graph.
2. Repeat the following a number of times.
3. Check if y is marked, and if not then choose one of its neighbors at random and set y to be that neighbor.

This may seem like a stupid algorithm, but it has certain advantages. For instance, it only need space $\mathcal{O}(\log N)$, because you need to keep track of the current vertex y (Say there are 8 vertices then they can be represented using 3 bits, thus $\log N$ space required to keep track of the current vertex), and maybe a counter that keeps track of how many steps you have already taken. Note that here we are assuming the neighbors of a given vertex are efficiently computable, so you don’t actually need to keep the whole graph in memory. This will be true for all graphs we consider here. Such an algorithm can for example decide whether there is a path from specific vertex y to a specific vertex x using $\mathcal{O}(\log N)$ space. We’d start the walk at y and only x would be marked; one can show that if there exists a path from y to x in G , then we will reach x in $\text{poly}(N)$ steps.

let us restrict attention to d -regular graphs without self-loops, so each vertex has exactly d neighbors. A random walk on such a graph G corresponds to an $N \times N$ symmetric matrix P , where $P_{x,y} = 1/d$ if (x,y) is an edge in G , and $P_{x,y} = 0$

otherwise. This P is the normalized adjacency matrix of G . If $v \in \mathbb{R}^N$ is a vector with a 1 at position y and 0s elsewhere, then Pv is a vector whose x -th entry is $(Pv)_x = 1/d$ if (x, y) is an edge, and $(Pv)_x = 0$ otherwise. In other words, Pv is the uniform probability distribution over the neighbors of y , which is what you get by taking one step of the random walk starting at y . More generally, if v is a probability distribution on the vertices, then Pv is the new probability distribution on the vertices after taking one step of the random walk, and $P^k v$ is the probability distribution after taking k steps.

Suppose we start with some probability-distribution vector v (which may or may not be centered at one vertex y). We will assume G is connected and not bipartite. Then $P^k v$ will converge to the uniform distribution over all vertices, and the speed of convergence is determined by the “gap” between the first eigenvalue of P and all the other eigenvalues. This can be seen as follows. Let $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_N$ be the eigenvalues of P , ordered by size, and v_1, v_2, \dots, v_N be corresponding orthogonal eigenvectors. (Analysing graphs by looking at the eigenvalues and eigenvectors of their adjacency matrix is called “algebraic graph theory” or “spectral graph theory”. The largest eigenvalue is $\lambda_1 = 1$, and corresponds to eigenvector $v_1 = u = (1/N)$ which is the uniform distribution over all vertices. One can show that our assumption that G is connected implies $\lambda_2 < 1$, and our assumption that G is not bipartite implies $\lambda_N > -1$. Hence all eigenvalues $\lambda_i \forall i \in \{2, \dots, N\}$ will be in $(-1, 1)$; the corresponding eigenvector v_i will be orthogonal to the uniform vector u , so the sum of its entries is 0. Let $\delta > 0$ be the difference between $\lambda_1 = 1$ and $\max_{i \geq 2} |\lambda_i|$ (hence $|\lambda_i| \leq 1 - \delta$ for all $i \geq 2$). This δ is called the “spectral gap” of the graph.

Now decompose the starting distribution v as $v = \sum_{i=1}^N \alpha_i v_i$. Since the sum of v 's entries is 1, and the sum of v_1 's entries is 1, while each other eigenvector v_i ($i \geq 2$) has entries summing to 0, it follows that $\alpha_1 = 1$. Now let us see what happens if we apply the random walk for k steps, starting from v :

$$P^k v = P^k \left(\sum_i \alpha_i v_i \right) = \sum_i \alpha_i \lambda_i^k v_i = u + \sum_{i \geq 2} \alpha_i \lambda_i^k v_i$$

Consider the squared norm of the difference between $P^k v$ and u :

$$\begin{aligned}\|P^k v - u\|^2 &= \left\| \sum_{i \geq 2} \alpha_i \lambda_i^k v_i \right\|^2 = \sum_{i \geq 2} |\alpha_i|^2 |\lambda_i|^{2k} \leq \sum_{i \geq 2} |\alpha_i|^2 (1 - \delta)^{2k} \\ &\leq (1 - \delta)^{2k} \sum_{i \geq 2} |\alpha_i|^2 \\ &\leq (1 - \delta)^{2k} \sum_i |\alpha_i|^2 \\ \|P^k v - u\|^2 &\leq \|v\|^2 (1 - \delta)^{2k}\end{aligned}$$

Since v is a probability distribution we have $\|v\|^2 \leq 1$. By choosing $k = \ln(1/\eta)/\delta$, we get $\|P^k v - u\| \leq \eta$. In particular, if δ is not too small, then we get quick convergence of the random walk to the uniform distribution u , no matter which distribution v started with (convergence in total variation distance can be derived from this by Cauchy-Schwarz, choosing $\eta \ll 1/\sqrt{N}$). Once we are close to the uniform distribution, we have probability roughly ϵ of hitting a marked vertex. Of course, the same happens if we just pick a vertex uniformly at random, but that may not always be an option if the graph is given implicitly.

Suppose it costs S to set up an initial state v ; it costs U to update the current vertex i.e. to perform one step of the random walk; and it costs C to check whether a given vertex is marked. “Cost” is left undefined for now, but typically it will count the number of queries to some input, or number of elementary operations. Consider a classical search algorithm that starts at v , and then repeats the following until it finds a marked vertex: check if the current vertex is marked, and if not run a random walk for roughly $1/\delta$ steps to get close to the uniform distribution. Ignoring constant factors, the expected cost before this procedure finds a marked item, is on the order of

$$S + \frac{1}{\epsilon} \left(C + \frac{1}{\delta} U \right)$$

22.2 Quantum walks

We will now modify the classical random walk algorithm to a quantum algorithm, where the distribution preserving matrix P is changed to a norm preserving matrix $W(P)$ (i.e. Unitary)

While the basis state of a classical random walk is the current vertex we are at, a basis state of a quantum walk has two registers, the first corresponding to the

current vertex and the second corresponding to the previous vertex. Equivalently, a basis state of a quantum walk corresponds to an edge of the graph.

Our resulting quantum walk algorithm for search will actually be quite analogous to Grover's algorithm. We will call a basis state $|x\rangle|y\rangle$ "good" if x is a marked vertex, and "bad" otherwise. Define $|p_x\rangle = \sum_y \sqrt{P_{xy}}|y\rangle$ to be the uniform superposition over the neighbors of x . As for Grover, define "good" and "bad" states as the superpositions over good and bad basis states:

$$|G\rangle = \frac{1}{\sqrt{|M|}} \sum_{x \in M} |x\rangle|p_x\rangle \quad \text{and} \quad |B\rangle = \frac{1}{\sqrt{N - |M|}} \sum_{x \notin M} |x\rangle|p_x\rangle$$

where M denotes the set of marked vertices. Note that $|G\rangle$ is just uniform superposition over all edges (x, y) where the first coordinate is marked, and $|B\rangle$ is just the uniform superposition over all edges (x, y) where the first coordinate is not marked.

If $\epsilon = |M|/N$ and $\theta = \arcsin(\sqrt{\epsilon})$ then the uniform state over all edges can be written as

$$|U\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle|p_x\rangle = \sin(\theta)|G\rangle + \cos(\theta)|B\rangle$$

Here is the algorithm for searching a marked vertex if an ϵ -fraction is marked. As in grover, if we don't know then we just run the algorithm repeatedly with exponentially decreasing guesses for $\epsilon(1/2, 1/4, 1/8, \dots)$. If at the end we still haven't found a marked item, we will conclude that probably none exists.

1. Setup the starting state $|U\rangle$
2. Repeat the following $\mathcal{O}(1/\sqrt{\epsilon})$ times:
 - (a) Reflect through $|B\rangle$
 - (b) reflect through $|U\rangle$
3. Measure the first register and check that the resulting vertex is marked.

The description and analysis of the algorithm takes place in the 2-dimensional space spanned by $|G\rangle$ and $|B\rangle$. Assuming we know how to implement steps (a) and (b), the proof that this algorithm finds a marked vertex is the same as for Grover and for amplitude amplification. We start with $|U\rangle = \sin(\theta)|G\rangle + \cos(\theta)|B\rangle$. The two reflections (a) and (b) increase the angle from θ to 3θ , moving us towards the good state (similarly to the analysis for Grover, you can draw a 2-dimensional picture with axes $|B\rangle$ and $|G\rangle$ to see this). More generally, after k applications of (a) and (b) our state has become

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

Choosing $k \approx \frac{\pi}{4\theta} = \mathcal{O}(1/\sqrt{\epsilon})$, we will have $\sin((2k+1)\theta) \approx 1$, at which point measuring the first register will probably yield a vertex x that is marked. We now look more closely how to implement the two kinds of reflections.

- (a) **Reflect through $|B\rangle$.** Reflecting through $|B\rangle$ is relatively straightforward: we just have to “recognize” whether the first register contains a marked x , and put a -1 if so (note that this is really a reflection through the subspace spanned by the bad basis states, but restricted to the 2-dimensional subspace spanned by $|G\rangle$ and $|B\rangle$ that’s the same as a reflection through $|B\rangle$).
- (b) **Reflect through $|U\rangle$** This is where the quantum walk comes in. Let \mathcal{A} be the subspace span $\{|x\rangle|p_x\rangle\}$ and \mathcal{B} be span $\{|p_y\rangle|y\rangle\}$. Let $\text{ref}(\mathcal{A})$ and $\text{ref}(\mathcal{B})$ denote reflections through \mathcal{A} and \mathcal{B} respectively. Define $W(P) = \text{ref}(\mathcal{B})\text{ref}(\mathcal{A})$ to be the product of these two reflections. This is the unitary analogue of P . Suppose we are able to implement the following two operations (even in a controlled manner):
 - (a) $|x\rangle|0\rangle \rightarrow |x\rangle|p_x\rangle$
 - (b) $|0\rangle|y\rangle \rightarrow |p_y\rangle|y\rangle$

Since (1)and (2) prepare a uniform superposition over the neighbors of x and y , respectively, one can think of them as taking one classical walk step “in superposition”. Note that $\text{ref}(\mathcal{A})$ can be implemented by applying the inverse of (a), putting a minus if the second register is not $|0\rangle$, and applying (a). We can similarly implement $\text{ref}(\mathcal{B})$ using (2) and its inverse. Here we can think $W(P) = \text{ref}(\mathcal{B})\text{ref}(\mathcal{A})$ as corresponding to four steps of the classical walk in superposition.

To see how to implement the reflection through $|U\rangle$, let us consider the eigenvalues and eigenvectors of $W(P)$. The eigenvalue of $W(P)$ can be related to the eigenvalues of $\lambda_1, \lambda_2, \dots$ of P as follows. Let $\theta_j \in [0, \pi/2]$ be such that $|\lambda_j| = \cos(\theta_j)$. We won’t prove it here, but it turns out that the eigenvalues of $W(P)$ are of the form $e^{\pm 2i\theta_j}$ where $\theta_j \geq \sqrt{2\delta}$, because $1 - \delta \geq |\lambda_j| = \cos(\theta_j) \geq 1 - \theta_j^2/2$. We want to implement a reflection $R(P)$ through the subspace spanned by the eigenvalue-1 eigenvectors of $W(P)$; restricted $R(P)$ through subspace spanned by $|G\rangle$ and $|B\rangle$ this will b the desired reflection through $|U\rangle$.

22.2.1 Continuous-time quantum walk

We will now introduce continuous-time quantum walk as a natural analog of continuous-time classical random walk, and we'll see some examples how the two kinds of processes differ.

Random walks come in two flavors: discrete - and continuous-time random walk, so we consider this case first. Given a graph $G = (V, E)$, we define the continuous-time random walk on G as follows. Let A be the adjacency matrix of G , the $|V| \times |V|$ matrix with

$$A_{j,k} = \begin{cases} 1, & (j, k) \in E \\ 0, & (j, k) \notin E \end{cases}$$

for every pair $j, k \in V$. In particular, if we disallow self loops, then the diagonal of A is zero. There is another matrix associated with G that is nearly as important: the Laplacian of G , which has

$$L_{j,k} = \begin{cases} -\deg(j) & j = k \\ 1 & (j, k) \in E \\ 0 & \text{otherwise} \end{cases}$$

where $\deg(j)$ denotes the degree of vertex j . (The Laplacian is sometimes defined different than this -e.g. sometimes with the opposite sign. We use this definition because it makes L as a discrete approximation of the Laplacian operator ∇^2 in the continuum.)

The continuous-time random walk on G is defined as the solution of the differential equation

$$\frac{d}{dt} p_j(t) = \sum_{k \in V} L_{jk} p_k(t)$$

Here $p_j(t)$ denotes the probability associated with vertex j at time t . This can be viewed as a discrete analogy of the diffusion equation. Note that

$$\frac{d}{dt} \sum_{j \in V} p_j(t) = \sum_{j, k \in V} L_{jk} p_k(t) = 0$$

(since the columns of L sum to 0), which shows that an initially normalized distribution remains normalized: the evolution of the continuous-time random walk for any time t is a stochastic process. The solution of the differential equation can be given in closed form as

$$p(t) = e^{Lt} p(0)$$

Chapter 23

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 23.1: 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 $|\psi(\vec{\theta}^*)\rangle = U_A(\vec{\theta}^*) |0\rangle$.

23.1 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.

23.1.1 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* (Hermitian matrix), 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, λ_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 $|\psi\rangle$ will be:

$$\begin{aligned} \langle\psi|\hat{\mathcal{H}}|\psi\rangle &= \langle\psi|\left(\sum_{k=0}^{N-1} \lambda_k |\phi_k\rangle \langle k|\right)|\psi\rangle \\ &= \sum_{k=0}^{N-1} \lambda_k \langle\psi|\phi_k\rangle \langle\phi_k|\psi\rangle \\ &= \sum_{k=0}^{N-1} \lambda_k |\langle\phi_k|\psi\rangle|^2 \end{aligned}$$

If we take into account that $\lambda_0 \leq \lambda_k, \forall k$, we have:

$$\begin{aligned} \langle\psi|\hat{\mathcal{H}}|\psi\rangle &= \sum_{k=0}^{N-1} \lambda_k |\langle\phi_k|\psi\rangle|^2 \\ &\geq \sum_{k=0}^{N-1} \lambda_0 |\langle\phi_k|\psi\rangle|^2 \\ &= \lambda_0 \sum_{k=0}^{N-1} |\langle\phi_k|\psi\rangle|^2 \\ &= \lambda_0 \end{aligned}$$

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 λ_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 $|\psi(\vec{\theta}^*)\rangle = |\phi_0\rangle$

23.1.2 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 λ_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.

23.2 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.

23.2.1 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.

23.2.2 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 $|001\rangle = X_0|000\rangle$. In this case, our unitary operator $U_R = X_0$,

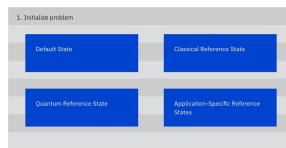


Figure 23.2: Reference state

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:

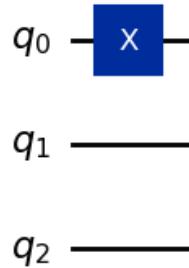


Figure 23.3: Classical Reference State Circuit

23.2.3 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 $|000\rangle$, one approach is to use a hadamard gate on qubit 0(H_0), a CNOT(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 23.4:

23.2.4 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
  
```

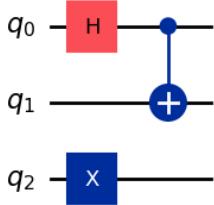


Figure 23.4: Quantum Reference State

```

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:

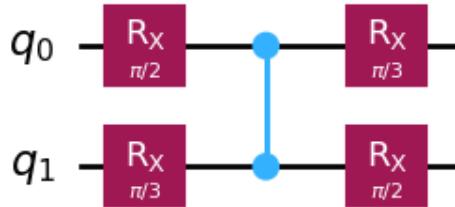


Figure 23.5: Reference State using Template

23.2.5 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(
    data)
6 zz_feature_map_reference.decompose().draw("mpl")

```

This outputs the following circuit:

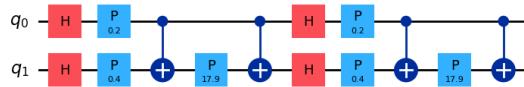


Figure 23.6: Reference State using ZZFeatureMap

23.3 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.

23.3.1 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

```

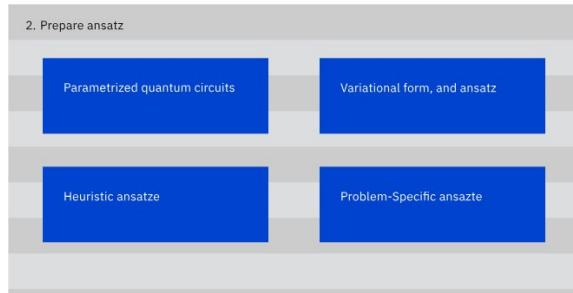


Figure 23.7: Prepare Ansatz

```

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:

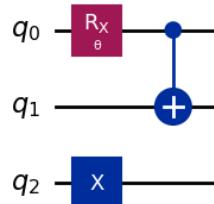


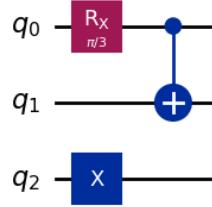
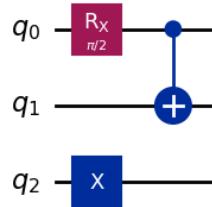
Figure 23.8: Parameterized Quantum Circuit

```

1 from math import pi
2
3 angle_list=[pi/3, pi/2]
4 circuits = [qc.assign_parameters({theta:angle} for angle in angle_list
5   )]
6
7 for circuit in circuits:
8   display(circuit.draw("mpl"))

```

This outputs the following circuits:

Figure 23.9: For $\theta = \pi/3$ Figure 23.10: For $\theta = \pi/2$

23.3.2 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: $U_A(\vec{\theta}) := U_V(\vec{\theta})U_R$.

Now, we have to construct an ansatz which represents a collection of parameterized states based on the different values of parameter vectors $\vec{\theta}$ we wish to explore

thus, creating a collection of parameterized quantum states $|\psi(\vec{\theta})\rangle$ to find the minima of the cost function $C(\vec{\theta}) = \langle\psi(\vec{\theta})|H|\psi(\vec{\theta})\rangle$. 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 $D = 2^{2n}$ because of the 2^n -complex entries of the n -dimensional Hilbert space requiring two real numbers each. 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 simulation 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),
6     CRZGate(theta)],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:

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

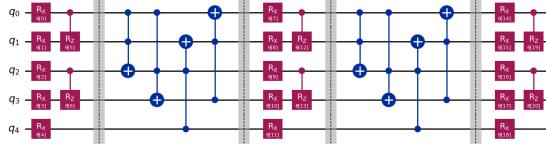


Figure 23.11: NLocal 5qubit example Circuit

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: In this case, we used the linear entanglement distribu-

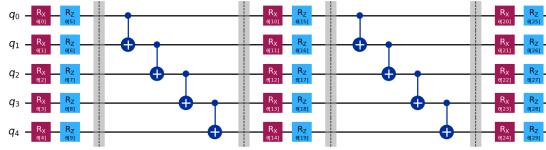


Figure 23.12: TwoLocal Ansatz Circuit Example

tion, 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=1)
4 ansatz.decompose().draw("mpl")

```

This results in following circuit:

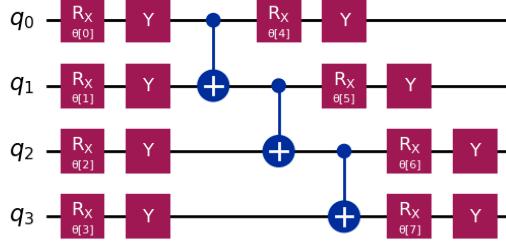


Figure 23.13: EfficientSU2 ansatze

23.3.3 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

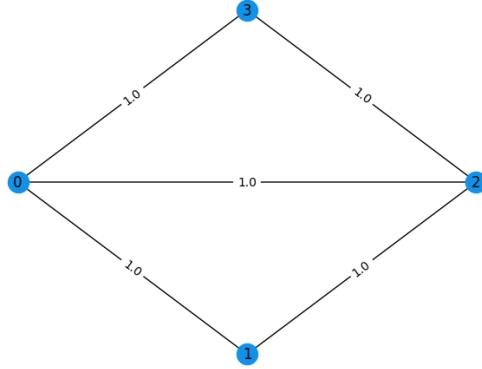


Figure 23.14: graph

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+ ZIIZ+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")

```

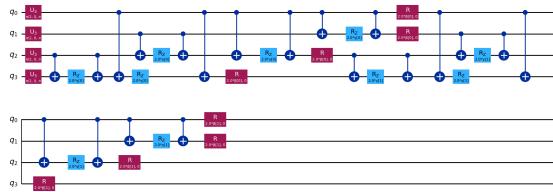


Figure 23.15: QAOA circuit

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 23.16: 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 maps based on parameterized quantum circuits that are hard to simulate classically is an important step towards obtaining a potential advantage over classical machine learning approaches and is an active area 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 (θ)

```
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(variation_form)
10 vqc_ansatz.decompose().draw("mpl")
```

The high-level variational workload looks as follows: 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.

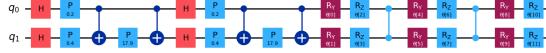


Figure 23.17: Circuit

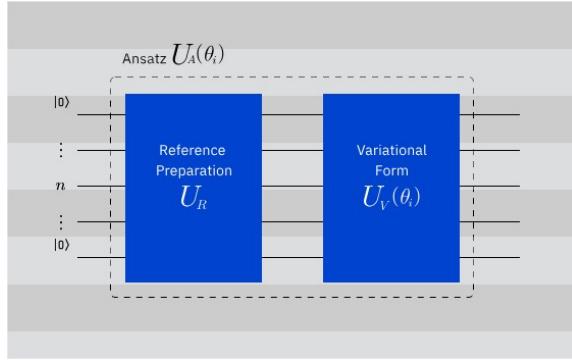


Figure 23.18: Variational workload

23.4 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

23.4.1 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.

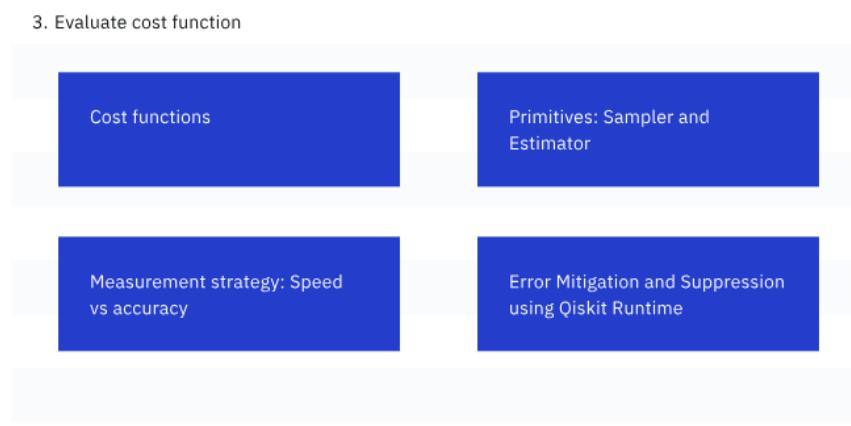


Figure 23.19: Cost function

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 (λ) 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 prepare 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 λ (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 λ_{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 : $\hat{H} = \sum_k^{Poly(n)} w_k \hat{P}_k$. The reader may notice the implicit assumption that probability sampling also needs to be efficient as explained for Sampler, which means

$$\langle H \rangle_\psi = \sum_k^{Poly(n)} w_k \sum_j^{Poly(n)} p_{kj} \lambda_{kj}$$

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 # and observable
6 original_circuit = QuantumCircuit(1)
7 original_circuit.h(0)
8
9 H = SparsePauliOp([ "X" , "Z" ] , [ 2 , -1 ])
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")

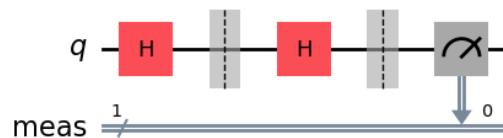
```



```

1 # Auxiliary circuit for X
2 aux_circuit[0].draw('mpl')

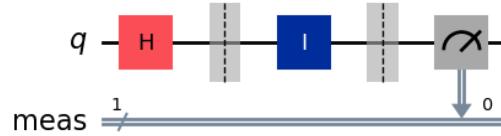
```



```

1 # Auxiliary circuit for Z
2 aux_circuits[1].draw('mpl')

```



We can now carry out the computation manually using Sampler and check the results on Estimator:

```

1 from qiskit.primitives import StatevectorSampler, StatevectorEstimator
2 from qiskit.result import QuasiDistribution
3 from qiskit.circuit.library import IGate, ZGate, XGate, YGate
4 import numpy as np
5
6
7 ## SAMPLER
8 shots = 10000
9 sampler = StatevectorSampler()
10 job = sampler.run(aux_circuits, shots = shots)
11 data_pub = job.result()[1].data
12 bitstrings = data_pub.meas.get_bitstrings()
13 counts = data_pub.meas.get_counts()
14 quasi_dist = QuasiDistribution({outcome: freq / shots for outcome, freq
15     in counts.items()})
16 expvals = []
17 for pauli in H.paulis():
18     val = 0
19
20     if str(pauli) == "I":
21         Lambda = IGate().to_matrix().real
22
23     if str(pauli) == "X":
24         Lambda = XGate().to_matrix().real
25         val += Lambda[0][1] * quasi_dist.get(1)
26         val += Lambda[1][0] * quasi_dist.get(0)
27
28     if str(pauli) == "Y":
29         Lambda = XGate().to_matrix().real
30         val += Lambda[0][1] * 1.j * quasi_dist.get(1)
31         val += Lambda[1][0] * -1.j * quasi_dist.get(0)

```

```

32
33     if str(pauli) == "Z":
34         Lambda = ZGate().to_matrix().real
35         val += Lambda[0][0] * quasi_dist.get(0)
36         val += Lambda[1][1] * quasi_dist.get(1)
37
38     expvals.append(val)
39
40 print("Sampler results:")
41 for (pauli, expval) in zip(H.paulis, expvals):
42     print(f" >> Expected value of {str(pauli)}: {expval:.5f}")
43
44 total_expval = np.sum(H.coeffs * expvals).real
45 print(f" >> Total expected value: {total_expval:.5f}")
46
47 ## ESTIMATOR
48 observables = [
49     *H.paulis, H
50 ] # Note: run for individual Paulis as well as full observable H
51
52 estimator = StatevectorEstimator()
53 job = estimator.run([(original_circuit, observables)])
54 estimator_expvals = job.result()[0].data.evs
55
56 print("Estimator results:")
57 for (obs, expval) in zip(observables, estimator_expvals):
58     if obs is not H:
59         print(f" >> Expected value of {str(obs)}: {expval:.5f}")
60     else:
61         print(f" >> Total expected value: {expval:.5f}")

```

Output:

```

1 Sampler results:
2   >> Expected value of X: 1.00000
3   >> Expected value of Z: 0.00860
4   >> Total expected value: 1.99140
5 Estimator results:
6   >> Expected value of X: 1.00000
7   >> Expected value of Z: 0.00000
8   >> Total expected value: 2.00000

```

Mathematical rigor

Expressing $|\psi\rangle$ with respect to the basis of eigen states of \hat{H} , $|\psi\rangle = \sum_{\lambda} a_{\lambda} |\lambda\rangle$, it follows:

$$\begin{aligned}
 \langle \psi | \hat{H} | \psi \rangle &= \left(\sum_{\lambda'} a_{\lambda'}^* \langle \lambda' | \right) \hat{H} \left(\sum_{\lambda} a_{\lambda} |\lambda\rangle \right) \\
 &= \sum_{\lambda} \sum_{\lambda'} a_{\lambda'}^* a_{\lambda} \langle \lambda' | \hat{H} | \lambda \rangle \\
 &= \sum_{\lambda} \sum_{\lambda'} a_{\lambda'}^* a_{\lambda} \lambda \langle \lambda' | \lambda \rangle \\
 &= \sum_{\lambda} -\lambda \sum_{\lambda'} a_{\lambda'}^* a_{\lambda} \lambda \delta_{\lambda, \lambda'} \\
 &= \sum_{\lambda} |a_{\lambda}|^2 \lambda \\
 &= \sum_{\lambda} p_{\lambda} \lambda
 \end{aligned}$$

Since we do not know the eigenvalue or eigenstate of the target observable \hat{H} , first we need to consider its diagonalization. given that \hat{H} is hermitian, there exists a unitary transformation such that $\hat{H} = V^{\dagger} \Lambda V^{\dagger}$, where Λ is the diagonal eigenvalue matrix, so $\langle j | \lambda | k \rangle = 0$ if $j \neq k$, and $\langle j | \Lambda | j \rangle = \lambda_j$.

This implies that the expected value can be rewritten as:

$$\begin{aligned}
 \langle \psi | \hat{H} | \psi \rangle &= \langle \psi | V^{\dagger} \Lambda V | \psi \rangle \\
 &= \langle \psi | V^{\dagger} \left(\sum_{j=0}^{2^n-1} |j\rangle \langle j| \right) \Lambda \left(\sum_{k=0}^{2^n-1} |k\rangle \langle k| \right) V | \psi \rangle \\
 &= \sum_{j=0}^{2^n-1} \sum_{k=0}^{2^n-1} \langle \psi | V^{\dagger} | j \rangle \langle j | \Lambda | k \rangle \langle k | V | \psi \rangle \\
 &= \sum_{j=0}^{2^n-1} \langle \psi | V^{\dagger} | j \rangle \langle j | \Lambda | j \rangle \langle j | V | \psi \rangle \\
 &= \sum_{j=0}^{2^n-1} | \langle j | V | \psi \rangle |^2 \lambda_j
 \end{aligned}$$

Given that if a system is in the state $| \phi \rangle = V | \psi \rangle$ the probability of measuring $| j \rangle$ is $p_j = | \langle j | \phi \rangle |^2$, the above expected value can be expressed as:

$$\langle \psi | \hat{H} | \psi \rangle = \sum_{j=0}^{2^n-1} p_j \lambda_j$$

It is very important to note that the probabilities are taken from the state $V | \psi \rangle$ instead of $| \psi \rangle$. This is why the matrix V is absolutely necessary.

One might be wondering how to obtain the matrix V and the eigenvalues Λ . If you already had the eigenvalues, then there would be no need to use a quantum computer since the goal of variational algorithms is to find these eigenvalues of \hat{H} .

Fortunately, there is a way around that: any $2^n \times 2^n$ matrix can be written as a linear combination of 4^n tensor products of n Pauli matrices and identities, all of which are both Hermitian and Unitary with known V and Λ . This is what Runtime's **Estimator** does internally by decomposing any Operator object into a SparsePauliOp.

Here the Operators that can be used:

Operator	σ	V	Λ
I	$\sigma_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$V_0 = I$	$\Lambda_0 = I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
X	$\sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$V_1 = H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$\Lambda_1 = \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Y	$\sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$V_2 = HS^\dagger = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ 1 & i \end{pmatrix}$	$\Lambda_2 = \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Z	$\sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$V_3 = I$	$\Lambda_3 = \sigma_3 \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

Table 23.1: Operators that can be used

So let's rewrite \hat{H} with respect to the Paulis and identities:

$$\hat{H} = \sum_{k_{n-1}=0}^3 \dots \sum_{k_0=0}^3 w_{k_{n-1} \dots k_0} \sigma_{k_{n-1}} \otimes \dots \otimes \sigma_{k_0} = \sum_{k=0}^{4^n-1} w_k \hat{P}_k$$

where $k = \sum_{l=0}^{n-1} 4^l k_l \equiv k_{n-1} \dots k_0$ for $k_{n-1}, \dots, k_0 \in \{0, 1, 2, 3\}$ (i.e.base 4), and

$$\hat{P}_k := \sigma_{k_{n-1}} \otimes \dots \otimes \sigma_{k_0};$$

$$\begin{aligned} \langle \psi | \hat{H} | \psi \rangle &= \sum_{k=0}^{4^n-1} w_k \sum_{j=0}^{2^n-1} |\langle j | V_k | \psi \rangle|^2 \langle j | \Lambda_k | j \rangle \\ &= \sum_{k=0}^{4^n-1} w_k \sum_{j=0}^{2^n-1} p_{kj} \lambda_{kj} \end{aligned}$$

where $V_k = V_{k_{n-1}} \otimes \dots \otimes V_{k_0}$ and $\Lambda_k := \Lambda_{k_{n-1}} \otimes \dots \otimes \Lambda_{k_0}$, such that: $\hat{P}_k = V_k^\dagger \Lambda_k V_k$.

23.4.2 Cost functions

In general, cost functions are used to describe the goal of a problem and how well a trial state is performing with respect to that goal. This definition can be applied to various examples in chemistry, machine learning, finance, optimization, and so on.

Let's consider a simple example of finding the ground state of a system. Our objective is to minimize the expectation value of the observable representing energy (Hamiltonian $\hat{\mathcal{H}}$):

$$\min_{\vec{\theta}} \langle \psi(\vec{\theta}) | \hat{\mathcal{H}} | \psi(\vec{\theta}) \rangle$$

We can use the Estimator to evaluate the expectation value and pass this value to an optimizer to minimize. If the optimization is successful, it will return a set of optimal parameter values $\vec{\theta}^*$, from which we will be able to construct the proposed solution state $|\psi(\vec{\theta}^*)\rangle$ and compute the observed the expectation value as $C(\vec{\theta}^*)$.

Notice how we will only be able to minimize the cost function for the limited set of states that we are considering. This leads us to two separate possibilities:

- **Our ansatz does not define the solution state across the search pace:**
If this is the case, our optimizer will never find the solution, and we need to experiment with other ansatzes that might be able to represent our search space more accurately.
- **Our optimizer is unable to find this valid solution:** Optimization can be globally defined and locally defined.

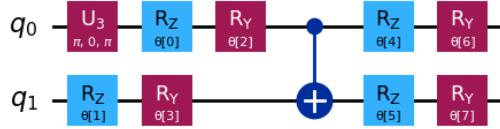
All in all, we will be performing a classical optimization loop but relying on the evaluation of the cost function to a quantum computer. From this perspective, one could think of the optimization as a purely classical endeavor where we call some black-box quantum oracle each time the optimizer needs to evaluate the cost function.

```

1 def cost_func_vqe(params, circuit, hamiltonian, estimator):
2     """Return estimate of energy from estimator
3
4     Parameters:
5         params (ndarray): Array of ansatz parameters
6         ansatz (QuantumCircuit): Parameterized ansatz circuit
7         hamiltonian (SparsePauliOp): Operator representation of
8             Hamiltonian
9         estimator (Estimator): Estimator primitive instance
10
11    Returns:
12        float: Energy estimate
13    """
14    pub = (circuit, hamiltonian, params)
15    cost = estimator.run([pub]).result()[0].data.evs
16
17 #    cost = estimator.run(ansatz, hamiltonian, parameter_values=params)
18 #.result().values[0]
19 return cost
20
21
22 from qiskit.circuit.library import TwoLocal
23
24 observable = SparsePauliOp.from_list([( "XX" , 1) , ( "YY" , -3)])
25
26 reference_circuit = QuantumCircuit(2)
27 reference_circuit.x(0)
28
29 variational_form = TwoLocal(
30     2,
31     rotation_blocks=[ "rz" , "ry" ] ,
32     entanglement_blocks="cx",
33     entanglement="linear",
34     reps=1,
35 )
36 ansatz = reference_circuit.compose(variational_form)
37
38 theta_list = (2 * np.pi * np.random.rand(1, 8)).tolist()
39 #ansatz.assign_parameters(theta_list)
40 #print(ansatz.num_parameters)
41 ansatz.decompose().draw('mpl')

```

We will first carry this out using a simulator: the StatevectorEstimator. This is usually advisable for debugging, but we will immediately follow the debugging run with a calculating on a real quantum hardware. increasingly, problem of interest are no longer classical simulable without state-of-the-art super computing facilities.



```

1 estimator = StatevectorEstimator()
2 cost = cost_func_vqe(theta_list, ansatz, observable, estimator)
3 print(cost)

1 [-1.14948565]

```

We will now proceed with running on real hardware. Note the syntax changes. The steps involving the pass_manager will be discussed further in the next example.

```

1 #Estimated usage: < 1 min. Benchmarked at 7 seconds on ibm_osaka,
  5–23–24
2 #Load necessary packages:
3
4 from qiskit_ibm_runtime import QiskitRuntimeService, Session,
    EstimatorV2 as Estimator
5 from qiskit.transpiler.preset_passmanagers import
    generate_preset_pass_manager
6
7 #Select the least busy backend:
8
9 service = QiskitRuntimeService(channel="ibm_quantum")
10 backend = service.least_busy(
11     operational=True, min_num_qubits=ansatz.num_qubits, simulator=False
12 )
13
14 #Use a pass manager to transpile the circuit and observable for the
  specific backend being used:
15
16 pm = generate_preset_pass_manager(backend=backend, optimization_level
  =1)
17 isa_ansatz = pm.run(ansatz)
18 isa_observable = observable.apply_layout(layout = isa_ansatz.layout)
19
20 #Open a Runtime session:
21
22 session = Session(backend=backend)
23
24 # Use estimator to get the expected values corresponding to the ansatz
25

```

```

26 estimator = Estimator(session=session)
27 cost = cost_func_vqe(theta_list, isa_ansatz, isa_observable, estimator)
28
29 # Close session after done
30 session.close()
31 print(cost)

1 [-0.79596413]

```

Note that the values obtained from the two calculations above are very similar. Techniques for improving results will be discussed further below.

Example mapping to non-physical systems

The maximum cut (Max-Cut) problem is a combinatorial optimization problem that involves dividing the vertices of a graph into two disjoint sets such that the number of edges between the two sets is maximized. More formally, given an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, the Max-Cut problem asks to partition the vertices into two disjoint subsets, S and T , such that the number of edges with one endpoint S and the other in T is maximized.

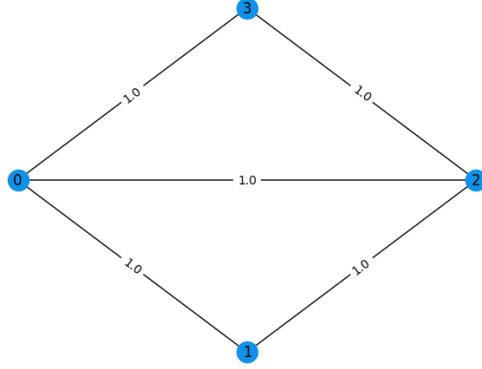
We can apply Max-Cut to solve a various problems including: clustering, network design, phase transitions, etc. We'll start by creating a problem graph:

```

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")

```

This problem can be expressed as a binary optimization problem. For each node $0 \leq i < n$, where n is the number of nodes of the graph (in this case $n = 4$), we will consider the binary variable x_i . This variable will have the value 1 if node i is one of the groups that we'll label 1 and 0 if it's in the other group, that we'll label as 0. We will also denote as w_{ij} (element (i, j) of the adjacency matrix w) the weight of the edge that goes from node i to node j . because the graph is undirected, $w_{ij} = w_{ji}$.



Then we can formulate our problem as maximizing the following cost function:

$$\begin{aligned}
 C(\vec{x}) &= \sum_{i,j=0}^n w_{ij}x_i(1 - x_j) \\
 &= \sum_{i,j=0}^n w_{ij}x_i - \sum_{i,j=0}^n w_{ij}x_i x_j \\
 &= \sum_{i,j=0}^n w_{ij}x_i - \sum_{i=0}^n \sum_{j=0}^i 2w_{ij}x_i x_j
 \end{aligned}$$

To solve this problem with a quantum computer, we are going to express the cost function as the expected value of an observable. However, the observables that Qiskit admits natively consist of Pauli operators, that have eigenvalues 1 and -1 instead of 0 and 1. That's why we are going to make the following change of variable:

Where $\vec{x} = (x_0, x_1, \dots, x_{n-1})$. We can use the adjacency matrix w to comfortably access the weights of all the edge. This will be used to obtain out cost function:

$$z_i = 1 - 2x_i \rightarrow x_i = \frac{1 - z_i}{2}$$

This implies that:

$$x_i = 0 \rightarrow z_i = 1$$

$$x_i = 1 \rightarrow z_i = -1$$

So the new cost function we want to maximize is:

$$\begin{aligned} C(\vec{z}) &= \sum_{i,j=0}^n w_{ij} \left(\frac{1 - z_i}{2} \right) \left(1 - \frac{1 - z_j}{2} \right) \\ &= \sum_{i,j=0}^n \frac{w_{ij}}{4} - \sum_{i,j=0}^n \frac{w_{ij}}{4} z_i z_j \\ &= \sum_{i=0}^n \sum_{j=0}^i \frac{w_{ij}}{2} - \sum_{i=0}^n \sum_{j=0}^i \frac{w_{ij}}{2} z_i z_j \end{aligned}$$

Moreover, the natural tendency of a quantum computer is to find minima (usually the lowest energy) instead of maxima so instead of maximizing $X(\vec{z})$ we are going to minimize:

$$-C(\vec{z}) = \sum_{i=0}^n \sum_{j=0}^i \frac{w_{ij}}{2} z_i z_j - \sum_{i=0}^n \sum_{j=0}^i \frac{w_{ij}}{2}$$

Now that we have a cost function to minimize whose variables can have the values -1 and 1, we can make the following analogy with the Pauli Z:

$$z_i \equiv Z_i = I \otimes \dots \otimes Z \otimes \dots \otimes I$$

with Z at the i th position and I everywhere else.

In other words, the variable z_i will be equivalent to a Z gate acting on qubit i . Moreover:

$$Z_i |x_{n-1} \dots x_0\rangle = z_i |x_{n-1} \dots x_0\rangle \rightarrow \langle x_{n-1} \dots x_0 | Z_i |x_{n-1} \dots x_0\rangle = z_i$$

Then the observable we are going to consider is:

$$\hat{H} = \sum_{i=0}^n \sum_{j=0}^i \frac{w_{ij}}{2} Z_i Z_j$$

to which we will have to add the independent term afterwards:

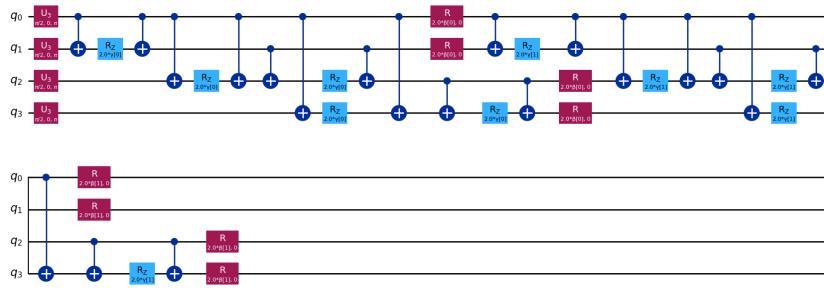
$$\text{offset} = - \sum_{i=0}^n \sum_{j=0}^i \frac{w_{ij}}{2}$$

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): $IIZZ + IZIZ + IZZI + ZIIZ + ZZII$. Once the operator is constructed, the ansatz for the QAOA algorithm can easily be built by using the QAOAAnsatz circuit from the Qiskit circuit library.

```

1 from qiskit.circuit.library import QAOAAnsatz
2 from qiskit.quantum_info import SparsePauliOp
3
4 hamiltonian = SparsePauliOp.from_list([
5     ("IIZZ", 1), ("IZIZ", 1), ("IZZI", 1), ("ZIIZ", 1), ("ZZII", 1)
6 ])
7
8
9 ansatz = QAOAAnsatz(hamiltonian, reps=2)
10 # Draw
11 ansatz.decompose(reps=3).draw("mpl")

```



```

1 # Sum the weights , and divide by 2
2
3 offset = - sum(edge[2] for edge in edges) / 2
4 print(f"""Offset: {offset}""")

```

1 Offset: -2.5

With the Runtime Estimator directly taking a Hamiltonian and parameterized ansatz, and returning the necessary energy, The cost function for a QAOA instance is quite simple:

```

1 def cost_func(params, ansatz, hamiltonian, estimator):
2     """Return estimate of energy from estimator
3
4     Parameters:
5         params (ndarray): Array of ansatz parameters
6         ansatz (QuantumCircuit): Parameterized ansatz circuit
7         hamiltonian (SparsePauliOp): Operator representation of
8             Hamiltonian
9         estimator (Estimator): Estimator primitive instance
10
11    Returns:
12        float: Energy estimate

```

```

12 """
13     pub = (ansatz, hamiltonian, params)
14     cost = estimator.run([pub]).result()[0].data.evs
15 #     cost = estimator.run(ansatz, hamiltonian, parameter_values=params)
16 #         .result().values[0]
17     return cost
18
19 import numpy as np
20
21 x0 = 2 * np.pi * np.random.rand(ansatz.num_parameters)
22
23 estimator = StatevectorEstimator()
24 cost = cost_func_vqe(x0, ansatz, hamiltonian, estimator)
25 print(cost)
26
27 3.1377748463972157
28
29 #Estimated usage: < 1 min, benchmarked at 6 seconds on ibm_osaka,
30 #      5-23-24
31 #Load some necessary packages:
32
33 from qiskit_ibm_runtime import QiskitRuntimeService
34 from qiskit_ibm_runtime import Session, EstimatorV2 as Estimator
35
36 #Select the least busy backend:
37
38 backend = service.least_busy(
39     operational=True, min_num_qubits=ansatz.num_qubits, simulator=False
40 )
41
42 #Use a pass manager to transpile the circuit and observable for the
43 #specific backend being used:
44
45 pm = generate_preset_pass_manager(backend=backend, optimization_level
46 =1)
47 isa_ansatz = pm.run(ansatz)
48 isa_hamiltonian = hamiltonian.apply_layout(layout = isa_ansatz.layout)
49
50 #Open a Runtime session:
51
52 session = Session(backend=backend)
53
54 # Use estimator to get the expected values corresponding to each ansatz
55 estimator = Estimator(session=session)
56
57 cost = cost_func_vqe(x0, isa_ansatz, isa_hamiltonian, estimator)

```

```

27
28 # Close session after done
29 session.close()
30 print(cost)

```

```

1 1.8429589920081832

```

We will revisit this example in Applications to explore how to leverage an optimizer to iterate through the search space. Generally, speaking, this includes:

- Leveraging an optimizer to find optimal parameters
- Binding optimal parameters to the ansatz to find the eigenvalues
- Translating the eigenvalues to our problem definition

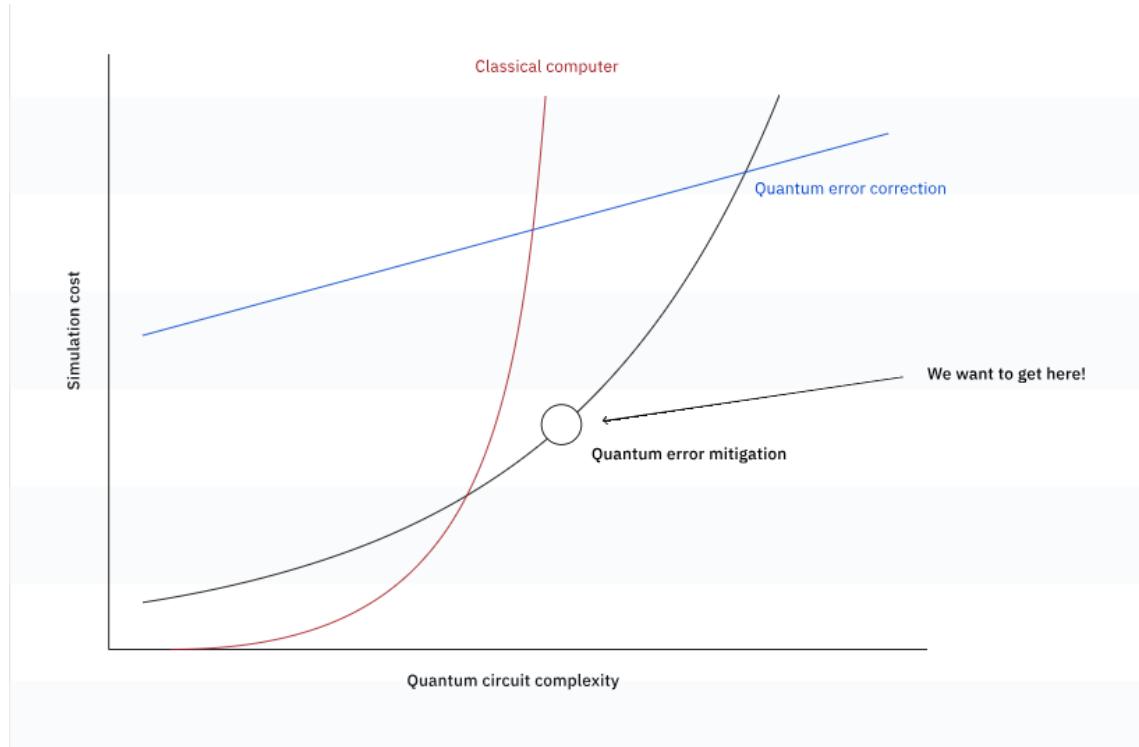
Measurement Strategy: speed vs accuracy

As mentioned, we are using noisy quantum computer as a black box oracle, where noise can make the retrieved values non-deterministic, leading to random fluctuations which, in turn, will harm - or even completely prevent the convergence of certain optimizers to a proposed solution. This is a general problem that we must address as we incrementally explore quantum utility and progress towards quantum advantage: We can use Qiskit Runtime Primitive's error suppression and error mitigation options to address noise and maximize the utility of today's quantum computers.

Error Suppression

Error suppression refers to techniques used to optimize and transform a circuit during compilation in order to minimize errors. This is a basic error handling technique that usually results in some classical pre-processing overhead to the overall runtime. The overhead includes transpiling circuits to run on a quantum hardware by:

- Expressing the circuit using the native gates available on a quantum system.
- Mapping the virtual qubits to physical qubits
- Adding SWAPs based on connectivity requirements
- Optimizing 1Q and 2Q gates
- Adding dynamical decoupling to idle qubits to prevent the effects of decoherence.



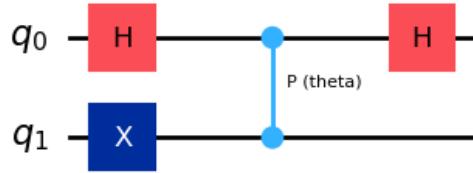
Primitives allow for the use of error suppression techniques by setting the optimization_level option and selecting advanced transpilation options. Later, we will delve into different circuit construction methods to improve results, but for most cases, we recommend setting optimization_level=3.

We will visualize the value of increasing optimization in the transpilation process by looking at an example circuit with simple ideal behaviour.

```

1 from qiskit.circuit import Parameter, QuantumCircuit
2 from qiskit.quantum_info import SparsePauliOp
3
4 theta = Parameter("theta")
5
6 qc = QuantumCircuit(2)
7 qc.x(1)
8 qc.h(0)
9 qc.cp(theta, 0, 1)
10 qc.h(0)
11 observables = SparsePauliOp.from_list([('ZZ', 1)])
12
13 qc.draw("mpl")

```



The circuit above can yield sinusoidal expectation values of the observable given, provided we insert phases spanning an appropriate interval, such as $[0, 2\pi]$.

```

1 ## Setup phases
2 import numpy as np
3
4 phases = np.linspace(0, 2*np.pi, 50)
5
6 # phases need to be expressed as a list of lists in order to work
7 individual_phases = [[phase] for phase in phases]
  
```

We can use a simulator to show the usefulness of an optimized transpilation. We will return below to using real hardware to demonstrate the usefulness of error mitigation. We will use QiskitRuntime Service to get a real backend (in this case, ibm_kyoto), and use AerSimulator to simulate that backend, including its noise behavior.

```

1 from qiskit_ibm_runtime import QiskitRuntimeService
2 from qiskit_aer import AerSimulator
3
4 # get a real backend from the runtime service
5 service = QiskitRuntimeService()
6 backend = service.get_backend('ibm_sherbrooke')
7
8 # generate a simulator that mimics the real quantum system with the
9 # latest calibration results
9 backend_sim = AerSimulator.from_backend(backend)
  
```

We can now use a pass manager to transpile the circuit into the "instruction set architecture" or ISA of the backend. This is a new requirement in Qiskit Runtime: all circuits submitted to a backend must conform to the constraints of the backend's target, meaning they must be written in terms of the backend's ISA - i.e., the set of instructions the device can understand and execute. These target constraints are defined by factors like the device's native basis gates, its qubit connectivity, and - when relevant - its pulse and other instruction timing specifications.

Note that in the present case, we will do this twice: once with optimization_level=0, and once with it set to 3. Each time we will use the Estimator primitive to estimate the expectation values of the observable at different values of phase.

```

1 #Import estimator and specify that we are using the simulated backend:
2
3 from qiskit_ibm_runtime import EstimatorV2 as Estimator
4 estimator = Estimator(backend = backend_sim)
5
6 circuit = qc
7
8 #Use a pass manager to transpile the circuit and observable for the
9 #backend being simulated.
9 #Start with no optimization:
10
11 from qiskit.transpiler.preset_passmanagers import
12     generate_preset_pass_manager
13
14 pm = generate_preset_pass_manager(backend=backend_sim,
15                                     optimization_level=0)
14 isa_circuit = pm.run(circuit)
15 isa_observables = observables.apply_layout(layout = isa_circuit.layout)
16
17 noisy_exp_values = []
18 pub = (isa_circuit, isa_observables, [individual_phases])
19 cost = estimator.run([pub]).result()[0].data.evs
20 noisy_exp_values = cost[0]
21
22 #Repeat above steps, but now with optimization = 3:
23
24 exp_values_with_opt_es = []
25 pm = generate_preset_pass_manager(backend=backend_sim,
26                                     optimization_level=3)
26 isa_circuit = pm.run(circuit)
27 isa_observables = observables.apply_layout(layout = isa_circuit.layout)
28
29 pub = (isa_circuit, isa_observables, [individual_phases])
30 cost = estimator.run([pub]).result()[0].data.evs
31 exp_values_with_opt_es = cost[0]
```

Finally, we can plot the results, and we see that the precision of the calculation was fairly good even without optimization, but it definitely improved by increasing the optimization to level 3. Note that in deeper, more complicated circuits, the difference between optimization levels of 0 and 3 are likely to be more significant. This is a very simple circuit used as a toy model.

```

1 import matplotlib.pyplot as plt
2
3 plt.plot(phases, noisy_exp_values, "o", label="opt=0")
4 plt.plot(phases, exp_values_with_opt_es, "o", label="opt=3")
5 plt.plot(phases, 2 * np.sin(phases / 2) ** 2 - 1, label="ideal")
6 plt.ylabel("Expectation")
7 plt.legend()
8 plt.show()

```

Error Mitigation

Error mitigation refers to techniques that allow users to reduce circuit errors by modelling the device noise at the time of execution. Typically, this results in quantum pre-processing overhead related to model training and classical post-processing overhead to mitigate errors in the raw results by using the generated model.

The Qiskit Runtime primitive's resilience_level option specifies the amount of resilience to build against errors. Higher levels generate more accurate results at the expense of longer processing times due to quantum sampling overhead. Resilience levels can be used to configure the trade-off between cost and accuracy when applying error mitigation to your primitive query.

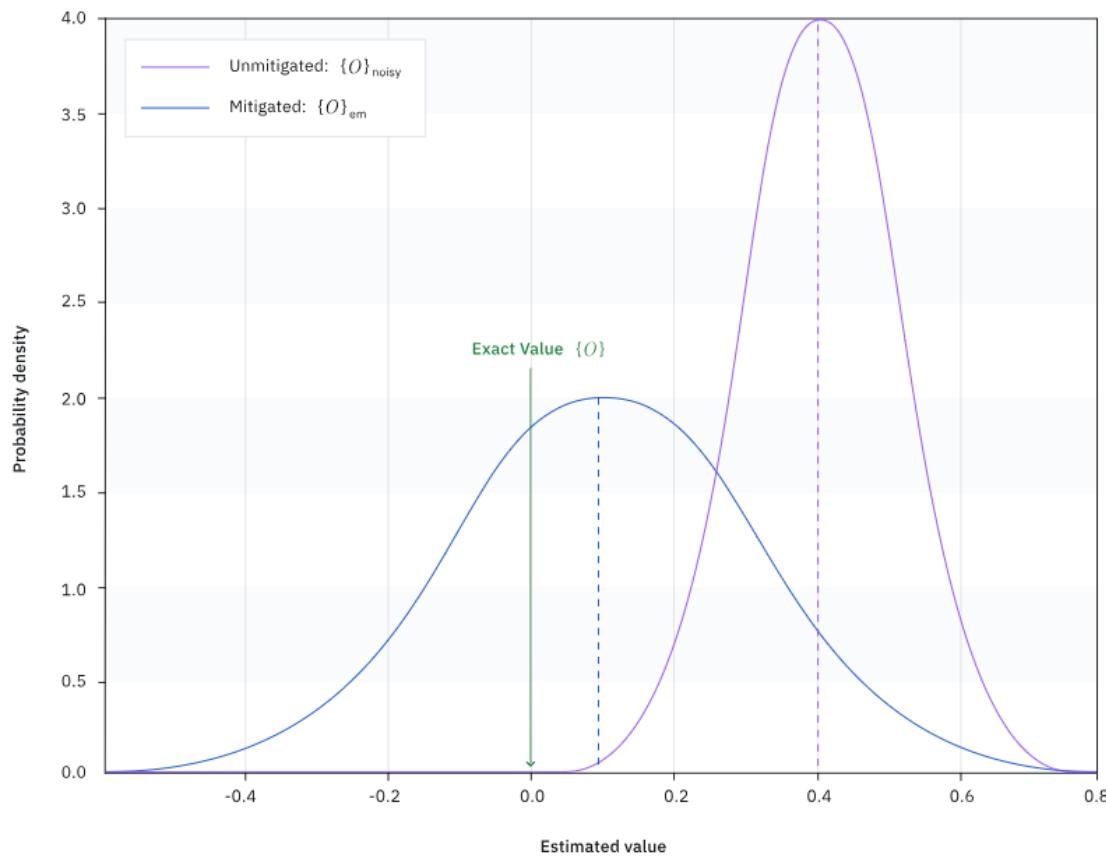
When implementing any error mitigation technique, we expect the bias in our results to be reduced with respect to the previous, unmitigated bias. In some cases, the bias may even disappear. However, this comes at a cost. As we reduce the bias in our estimated quantities, the statistical variability will increase (that is, variance), which can account for further increasing the number of shots per circuit in our sampling process. This will introduce overhead beyond that needed to reduce the bias, so it is not done by default. We can easily opt-in to this behavior by adjusting the number of shots per circuit in options.executions.shots, as shown in the example below.

We will now explore these error mitigation models at high level to illustrate the error mitigation that Qiskit Runtime primitives can perform without requiring full implementation details.

Twirled readout error extinction (T-Rex)

Twirled readout error extinction (T-Rex) uses a technique known as Pauli twirling to reduce the noise introduced during the process of quantum measurement. This technique assumes no specific form of noise, which makes it very general and effective.

Overall workflow:



1. Acquire data for the zero state with randomized bit flips (Pauli X before measurement)
2. Acquire data for the desired (noisy) state with randomized bit flips (Pauli X before measurement)
3. Compute the special function for each data set, and divide.



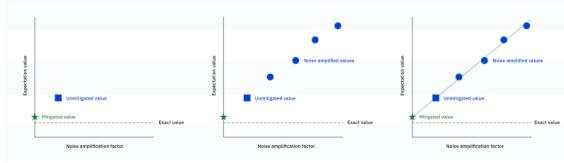
We can set this with options.resilience_level=1, demonstrated in the example below.

Zero noise extrapolation

Zero noise extrapolation (ZNE) works by first amplifying the noise in the circuit that is preparing the desired quantum state, obtaining measurements for several different levels of noise, and using those measurements to infer the noiseless result.

Overall workflow:

1. Amplify circuits noise for several noise factors.
2. Run every noise amplifies circuit
3. Extrapolate back to the zero noise limit.



we can set this with options.resilience_level=2. We can optimize this further by exploring a variety of noise_factors, noise_amplifiers, and extrapolators, but this is outside the scope of this. We encourage you to experiment with this options.

Each method comes with its own associated overhead: a trage-off between the number of quantum computations needed (time) and the accuracy of the results:

Methods	R=1, T-REx	R=2, ZNE
Assumptions	None	Ability to scale noise
Qubit overhead	1	1
Sampling overhead	2	$N_{noise-factors}$
Bias	0	$\mathcal{O}(\lambda_{noise-factors}^N)$

Table 23.2: Caption

Using Qiskit Runtime's mitigation and suppression options

Here's how to calculate an expectation value while using error mitigation and suppression in Qiskit Runtime. We can make use of precisely the same circuit and observable as before, but this time keeping the optimization level fixed at level 2, and now tuning the resilience or the error mitigation techniques beign used. This error mitigation process occurs multiple times throughout an optimization loop.

We perform this part on real hardware, since error mitigation is not available on simualtors.

```

1 # Estimated usage: 8 minutes , benchmarked on ibm_sherbrooke , 5-23-24
2
3 from qiskit_ibm_runtime import QiskitRuntimeService
4 from qiskit_ibm_runtime import Options, Session, EstimatorV2 as
   Estimator
5
6 # We select the least busy backend
7
8 backend = service.least_busy(
9     operational=True, min_num_qubits=ansatz.num_qubits, simulator=False
10)
11
12 #Initialize some variables to save the results from different runs:
13
14 exp_values_with_em0_es = []
15 exp_values_with_em1_es = []
16 exp_values_with_em2_es = []
17
18 #Use a pass manager to optimize the circuit and observables for the
   backend chosen:
19
20 pm = generate_preset_pass_manager(backend=backend, optimization_level
   =2)
21 isa_circuit = pm.run(circuit)
22 isa_observables = observables.apply_layout(layout = isa_circuit.layout)
23
24 # Open a session and run with no error mitigation:
25
26 with Session(backend=backend) as session:
27     session_options = Options()
28     session_options.execution.shots = 2000
29     session_options.resilience.level = 0
30
31     estimator = Estimator(session=session)
32     estimator.options.default_shots = 10_000

```

```

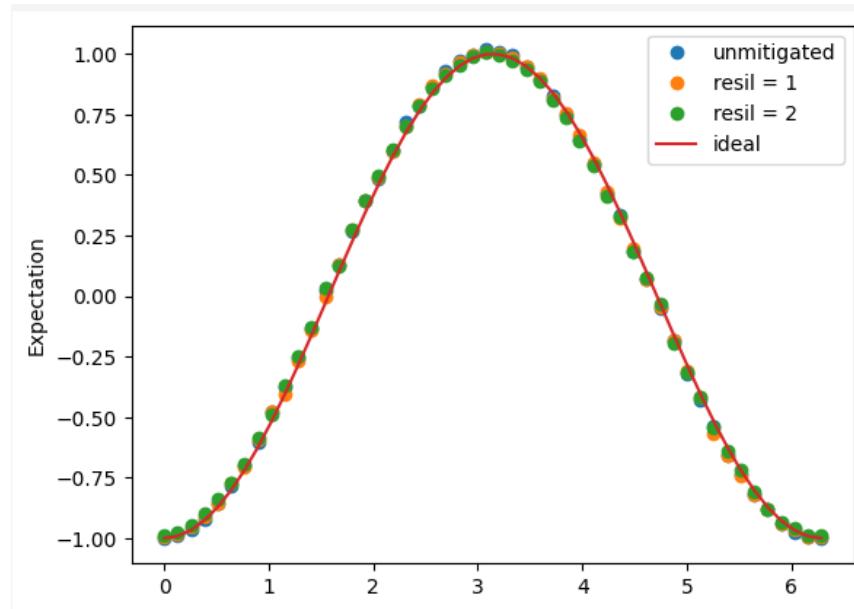
33
34     pub = (isa_circuit, isa_observables, [individual_phases])
35     cost = estimator.run([pub]).result()[0].data.evs
36
37 session.close()
38
39 exp_values_with_em0_es = cost[0]
40
41 # Open a session and run with resilience = 1:
42
43 with Session(backend=backend) as session:
44     session_options = Options()
45     session_options.execution.shots = 2000
46     session_options.resilience_level = 1
47
48 estimator = Estimator(session=session)
49 estimator.options.default_shots = 10_000
50
51 pub = (isa_circuit, isa_observables, [individual_phases])
52 cost = estimator.run([pub]).result()[0].data.evs
53
54 session.close()
55
56 exp_values_with_em1_es = cost[0]
57
58 # Open a session and run with resilience = 2:
59
60 with Session(backend=backend) as session:
61     session_options = Options()
62     session_options.execution.shots = 2000
63     session_options.resilience_level = 2
64
65 estimator = Estimator(session=session)
66 estimator.options.default_shots = 10_000
67
68 pub = (isa_circuit, isa_observables, [individual_phases])
69 cost = estimator.run([pub]).result()[0].data.evs
70
71 session.close()
72
73 exp_values_with_em2_es = cost[0]
```

As before, we can plot the resulting expectation values as a function of phase angles for the three levels of error mitigation used. With great difficulty, one can see that error mitigation improves the results slightly. Again, this effect is much more pronounced in deeper, more complicated circuits.

```

1 import matplotlib.pyplot as plt
2
3 plt.plot(phases, exp_values_with_em0_es, "o", label="unmitigated")
4 plt.plot(phases, exp_values_with_em1_es, "o", label="resil = 1")
5 plt.plot(phases, exp_values_with_em2_es, "o", label="resil = 2")
6 plt.plot(phases, 2 * np.sin(phases / 2) ** 2 - 1, label="ideal")
7 plt.ylabel("Expectation")
8 plt.legend()
9 plt.show()

```

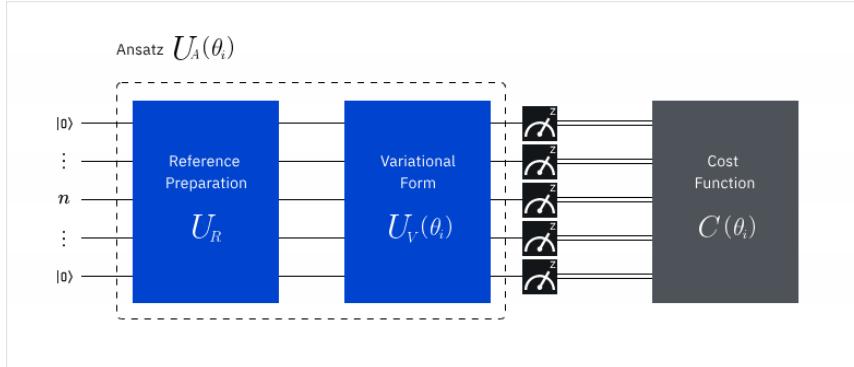


23.4.3 Summary

With this lesson, you learned how to create a cost function:

- Create a cost function
- How to leverage Qiskit Runtime primitives to mitigate and suppression noise
- how to define a measurement strategy to optimize speed vs accuracy

Here's high-level variational workload: Our cost function runs during every iteration of the optimization loop. The next lesson will explore how the classical optimizer uses our cost function evaluation to select new parameters.



```

1 import qiskit
2 import qiskit_ibm_runtime
3 print(qiskit.version.get_version_info())
4 print(qiskit_ibm_runtime.version.get_version_info())
1 1.2.0
2 0.28.0

```

23.5 Optimization Loops

Now we will see how to use an optimizer to iteratively explore the parameterized quantum states of our ansatz:

- Bootstrap an optimization loop
- understand trade-offs while using local and global optimizers
- Explore barren plateaus and how to avoid them

At a high level, optimizers are central to exploring our search space. the optimizer uses cost function evaluations to select the next set of parameters in a variational loop, and repeats the process until it reaches a stable state. At this stage, an optimal set of parameter values $\vec{\theta}^*$ is returned.

23.5.1 Local and Global Optimizers

We will first setup our problem before exploring each optimizer class:

4. Optimize parameters to get results

Bootstrapping Optimization

Local and Global Optimizers

Barren Plateaus

Gradient-Based and Gradient-Free Optimizers

```

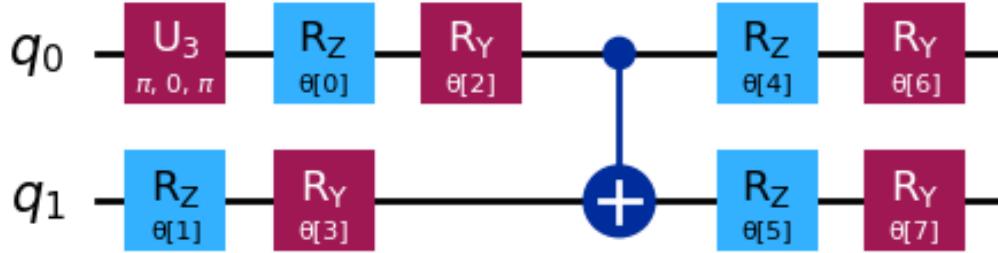
1 from qiskit import QuantumCircuit
2 from qiskit.quantum_info import SparsePauliOp
3 from qiskit.circuit.library import TwoLocal
4 import numpy as np
5
6 theta_list = (2 * np.pi * np.random.rand(1, 8)).tolist()
7 observable = SparsePauliOp.from_list([( "XX" , 1) , ( "YY" , -3)])
8
9 reference_circuit = QuantumCircuit(2)
10 reference_circuit.x(0)
11
12 variational_form = TwoLocal(
13     2 ,
14     rotation_blocks=[ "rz" , "ry" ] ,
15     entanglement_blocks="cx" ,
16     entanglement="linear" ,
17     reps=1 ,
18 )
19 ansatz = reference_circuit.compose(variationnal_form)
20
21 ansatz.decompose().draw( 'mpl' )

```

```

1 def cost_func_vqe(params , ansatz , hamiltonian , estimator):
2 """Return estimate of energy from estimator
3
4 Parameters:
5     params (ndarray): Array of ansatz parameters
6     ansatz (QuantumCircuit): Parameterized ansatz circuit
7     hamiltonian (SparsePauliOp): Operator representation of Hamiltonian
8     estimator (Estimator): Estimator primitive instance
9

```



```

10 Returns:
11     float: Energy estimate
12 """
13 pub = (ansatz, hamiltonian, params)
14 cost = estimator.run([pub]).result()[0].data.evs
15 return cost
1
2 from qiskit.primitives import StatevectorEstimator
3 estimator = StatevectorEstimator()

```

Local Optimizers

Local optimizers search for a point that minimizes the cost function starting at an initial point(s) $C(\vec{\theta}_0)$ and move to different points based on what they observe in the region they are currently evaluation on successive iterations. This implies that the convergence of these algorithms will usually be fast, but can be heavily dependent on the initial point. Local optimizers are unable to see beyond the region where they are evaluating and can be especially vulnerable to local minima, reporting convergence when they find one and ignoring other states with more favorable evaluations.

```

1 # SciPy minimizer routine
2 from scipy.optimize import minimize
3
4 x0 = np.ones(8)
5
6 result = minimize(cost_func_vqe, x0, args=(ansatz, observable,
7     estimator), method="SLSQP")
8 result

```

```

1 message: Optimization terminated successfully
2 success: True
3 status: 0
4     fun: -3.9999999645274
5         x: [ 1.000e+00  1.000e+00 -1.571e+00 -4.556e-05 -1.207e+00
6             -1.935e+00  4.079e-01 -4.079e-01]
7     nit: 12
8     jac: [ 1.192e-07  2.980e-08 -7.959e-04  2.543e-04  1.381e-03
9             1.381e-03  5.431e-04  5.431e-04]
10    nfev: 112
11    njev: 12

```

Global Optimizers

Global optimizers search for the point that minimizes the cost function over several regions of its domain (i.e. non-local), evaluating it iteratively (i.e. at iteration i) over a set of parameters vectors $\Theta_i = \vec{\theta}_{i,j} | \in \mathcal{J}_{opt}^i$ determined by the optimizer. This makes them less susceptible to local minima and somewhat independent of initialization, but also significantly slower to converge to a proposed solution.

Bootstrapping Optimization

Bootstrapping, or setting the initial value for parameters $\vec{\theta}$ based on a prior optimization, can help our optimizer converge on a solution faster. we refer to this as the initial point $\vec{\theta}_0$, and $|\psi(\vec{\theta}_0)\rangle = U_V(\vec{\theta}_0) |\rho\rangle$ as the initial state. This initial state differs from our reference state $|\rho\rangle$, as the former focuses on initial parameters set during our optimization loop, while the latter focuses on using known “reference” solutions. They may coincide if $U_V(\vec{\theta}_0) = I$ (i.e., the Identity operation).

When local optimizers converge to non-optimal local minima, we can try bootstrapping the optimization globally and refine the convergence locally. While this requires setting up two variational workloads, it allows the optimizer to find a more optimal solution than the local optimizer alone.

23.5.2 Gradient-Based and Gradient-Free Optimizers

Gradient-Based

For our cost function $C(\vec{\theta})$, if we have access to the gradient of the function $\vec{\nabla}C(\vec{\theta})$ starting from a initial point, the simplest way to minimize the function is to update the parameters towards the direction of steepest descent of the function. That is, we

update the parameters as $\vec{\theta}_{n+1} = \vec{\theta}_n - \eta \vec{\nabla} C(\vec{\theta})$, where η is the learning rate - a small, positive hyperparameter that controls the size of the update. We continue doing this until we converge to a local minimum of the cost function, $C(\vec{\theta}^*)$.

We can use this cost function and an optimizer to calculate optimal parameters.

```

1 # SciPy minimizer routine
2 from scipy.optimize import minimize
3
4 x0 = np.ones(8)
5
6 result = minimize(cost_func_vqe, x0, args=(ansatz, observable,
7 estimator), method="BFGS")
8
9 result
10
11 message: Optimization terminated successfully.
12 success: True
13 status: 0
14 fun: -3.999999999999782
15     x: [ 1.000e+00  1.000e+00  1.571e+00  2.785e-07  2.009e-01
16          -2.009e-01  6.342e-01 -6.342e-01]
17     nit: 14
18     jac: [ 5.960e-08 -2.980e-08  8.941e-07  1.013e-06  2.086e-07
19          1.490e-07  5.960e-08  5.960e-08]
20     hess_inv: [[ 1.000e+00 -2.999e-08 ... -7.780e-05 -6.011e-05]
21                 [-2.999e-08  1.000e+00 ... -3.475e-06 -7.574e-07]
22                 ...
23                 [-7.780e-05 -3.475e-06 ...  7.242e-01 -2.603e-01]
24                 [-6.011e-05 -7.574e-07 ... -2.603e-01  8.180e-01]]
25     nfev: 144
26     njev: 16

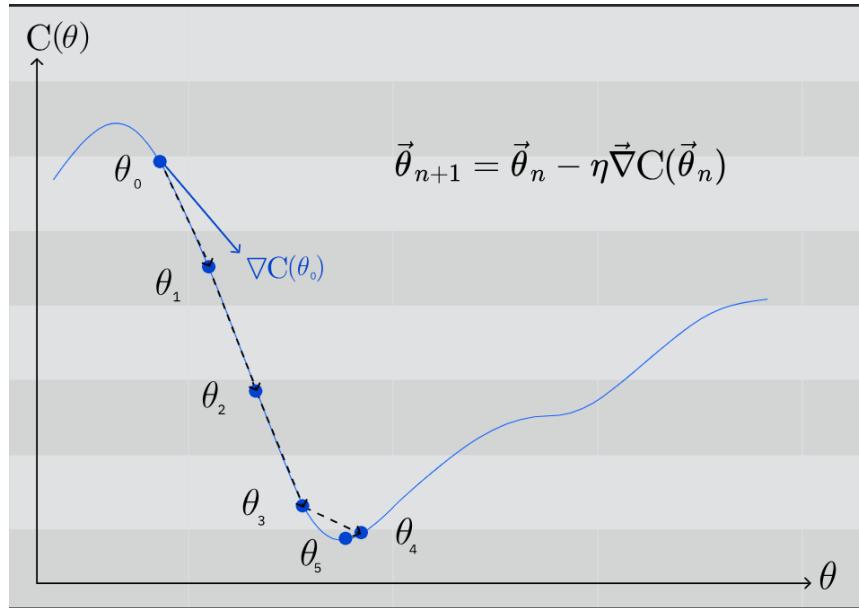
```

The main advantage of this type of optimization are the convergence speed, which can be very slow, and there is no guarantee to achieve the optimal solution.

Gradient-Free

Gradient-free optimization algorithms do not require gradient information and can be useful in situations where computing the gradient is difficult, expensive or too noisy. They also tend to be more robust in finding global optima, whereas gradient based methods tend to converge to local optima. We'll explore a few instance where a gradient-free optimizer can help avoid barren plateaus. However, gradient-free methods require higher computational resources, especially for problems with high-dimensional search spaces.

Here's an example that uses COBYLA optimizer instead:



```

1 # SciPy minimizer routine
2 from scipy.optimize import minimize
3
4 x0 = np.ones(8)
5
6 result = minimize(cost_func_vqe, x0, args=(ansatz, observable,
    estimator), method="COBYLA")
7
8 result

```

```

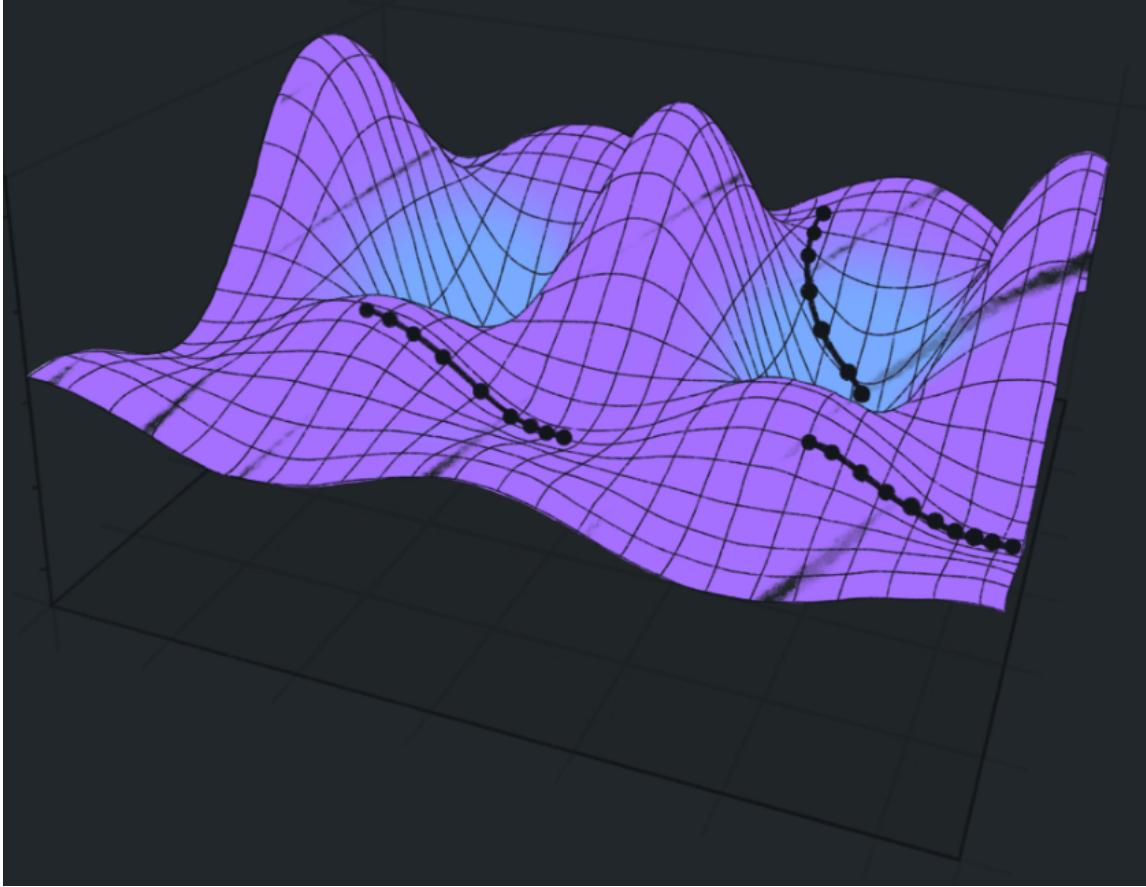
1 message: Optimization terminated successfully.
2 success: True
3 status: 1
4     fun: -3.999999832448676
5         x: [ 1.687e+00  1.250e+00  1.571e+00  3.142e+00  1.469e+00
6          -1.672e+00  1.556e+00  1.585e+00]
7     nfev: 126
8     maxcv: 0.0

```

23.5.3 Barren Plateaus

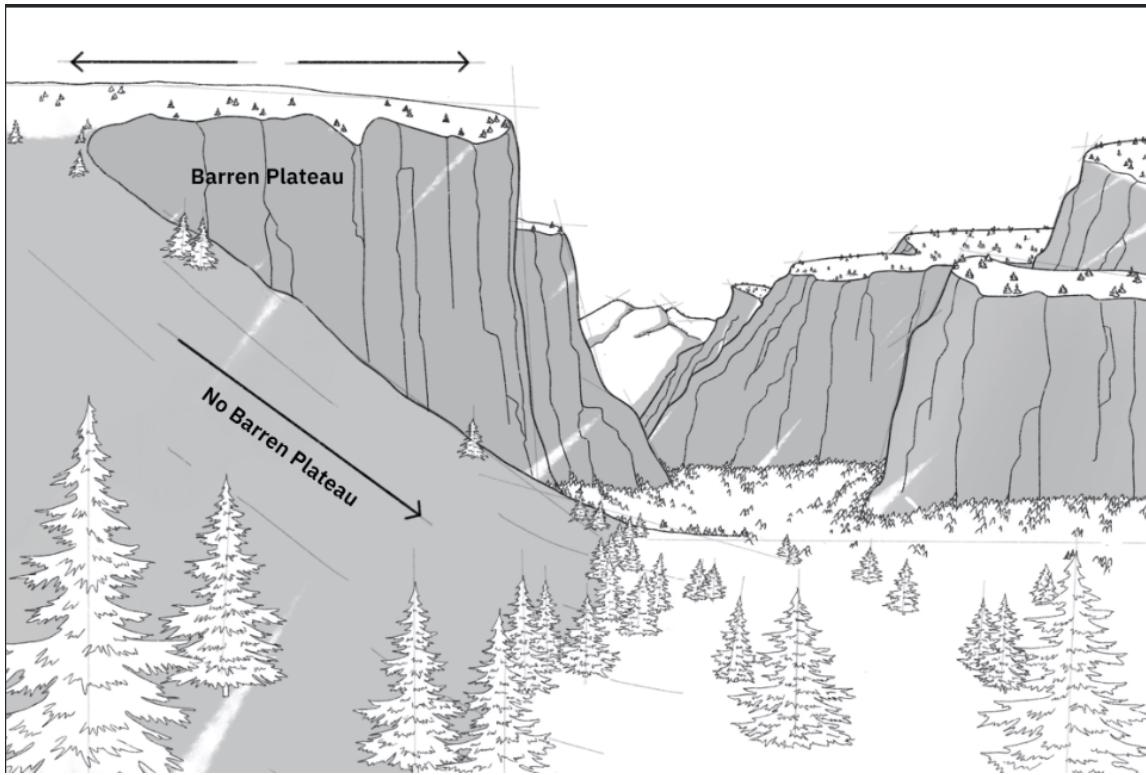
In fact, the cost landscape can be quite complicated, as shown by the hills and valleys in the example below. The optimization method navigates us around the cost landscape, search for the minimum, as shown by the black points and lines. We

can see two of the three searches end up in a local minimum of the landscape, rather than a global one. Regardless of the type of optimization method used, if the cost



landscape is relatively flat, it can be challenging for the method to determine the appropriate direction to search. This scenario is referred to as a barren plateaus, where the cost landscape becomes progressively flatter (and thus more challenging to determine the direction of the minimum). For a broad range of parameterized quantum circuits, the probability that the gradient along any reasonable direction is non-zero to some fixed precision decreases exponentially as the number of qubits increases. While this area is still under active research, we have a few recommendations to improve optimization performance:

- **Bootstrapping** can help the optimization loop avoid getting stuck in a parameter space where the gradient is small.



- **Experimenting with hardware-efficient ansatz:** Since we're using a noisy quantum system as a black box oracle, the quality of those evaluations can affect the optimizer's performance. Using hardware-efficient ansatz, such as EfficientSU2, may avoid producing exponentially small gradients.
- **Experimenting with error suppression and error mitigation:** the Qiskit Runtime primitives provide a simple interface to experiment with various optimization_level and resilience_setting respectively. This can be reduced the impact of noise and make the optimization process more efficient.
- **Experimenting with gradient-free optimizers:** Unlike gradient-based optimization algorithms, optimizers such as COBYLA do not rely on gradient information to optimize the parameters and are therefore less likely to be affected by the barren plateau.

23.5.4 Summary

With this lesson, you learned how to define your optimization loop:

- Bootstrap an optimization loop
- Understand trade-offs while using local and global optimizers
- Explore barren plateaus and how to avoid them

Our high level variational workload is complete:

Next, we'll explore specific variational algorithms with this framework in mind.

23.6 Instances and Extensions

This chapter will cover several quantum variational algorithms, including:

- Variational Quantum Eigensolver (VQE)
- Subspace Search VQE (SSVQE)
- Variational Quantum Deflation (VQD)
- Quantum Sampling Regression

By, using these algorithms, we will learn about several design ideas that can be incorporated into custom variational algorithms, such as weights, penalties, oversampling and undersampling. We encourage you to experiment with these concepts and share your findings with the community.

The Qiskit patterns framework applies to all these algorithms - but we will explicitly call out the steps only in the first example.

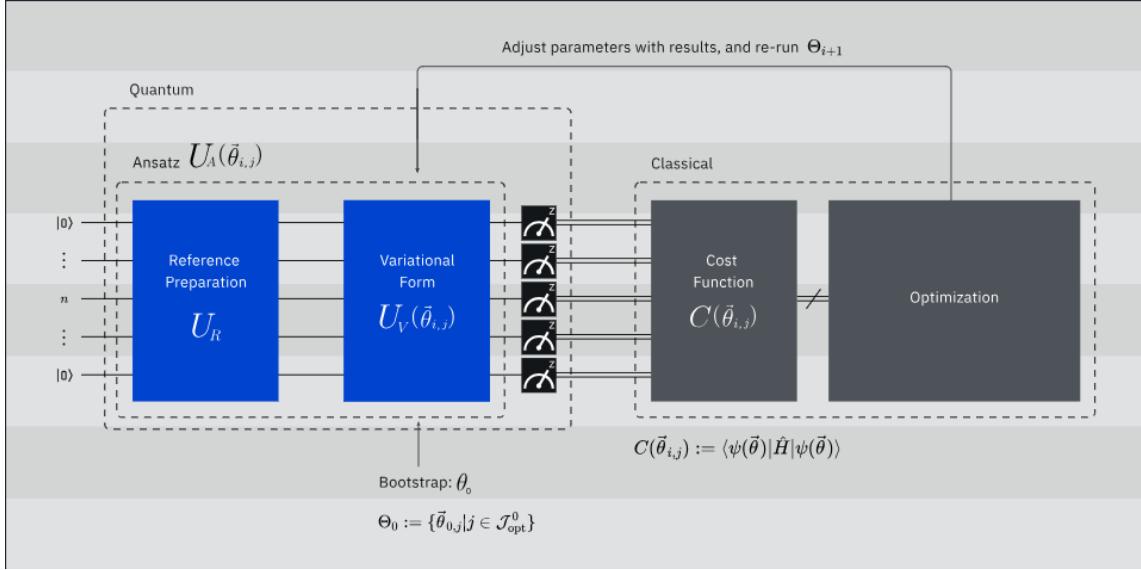
23.6.1 Variational Quantum eigensolver (VQE)

VQE is one of the most widely used variational quantum algorithms, setting up a template for other algorithms to build upon.

Step 1: Map classical inputs to a quantum problem

VQE's theoretical layout is simple:

- Prepare reference operators U_R
- We start from the state $|0\rangle$ and go to the reference state $|\rho\rangle$
- Apply the variational form $U_V(\vec{\theta}_{ij})$ to create an ansatz $U_A(\vec{\theta}_{ij})$
- We go from the state $|\rho\rangle$ to $U_V(\vec{\theta}_{ij})|\rho\rangle = |\psi(\vec{\theta}_{ij})\rangle$



- Bootstrap at $i = 0$ if we have a similar problem (typically found via classical simulation or sampling)
- Each optimizer will be bootstrapped differently, resulting in an initial set of parameter vectors $\Theta_0 = \vec{\theta}_{0,j} | j \in \mathcal{J}_{opt}^0$ (e.g., from an initial point $\vec{\theta}_0$).
- Evaluate the cost function $C(\vec{\theta}_{i,j}) = \langle\psi(\vec{\theta})|\hat{H}|\psi(\vec{\theta})\rangle$ for all prepared states on a quantum computer.
- Use a classical optimizer to select the next set of parameters Θ_{i+1} .
- Repeat the process until convergence is reached.

This is a simple classical optimization loop where we evaluate the cost function. Some optimizers may require multiple evaluations to calculate a gradient, determine the next iteration, or assess convergence.

Here's the example for the following observable:

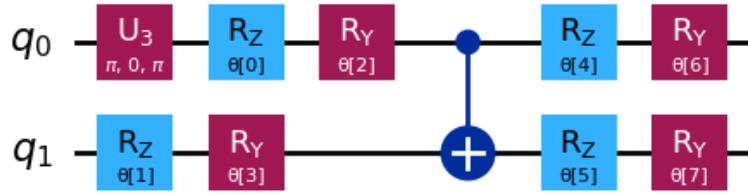
$$\hat{O}_1 = 2II - 2XX + 3YY - 3ZZ$$

VQE Implementation

```

1 from qiskit import QuantumCircuit
2 from qiskit.quantum_info import SparsePauliOp

```



```

3 from qiskit.circuit.library import TwoLocal
4 import numpy as np
5
6 theta_list = (2 * np.pi * np.random.rand(1, 8)).tolist()
7 observable=SparsePauliOp.from_list([( "II" , 2) , ( "XX" , -2) , ( "YY" , 3) , (
8   "ZZ" , -3)])
9
10 reference_circuit = QuantumCircuit(2)
11 reference_circuit.x(0)
12
13 variational_form = TwoLocal(
14   2,
15   rotation_blocks=[ "rz" , "ry" ] ,
16   entanglement_blocks="cx" ,
17   entanglement="linear" ,
18   reps=1,
19 )
20 ansatz = reference_circuit.compose(variational_form)
21 ansatz.decompose().draw( 'mpl' )

1 def cost_func_vqe(parameters , ansatz , hamiltonian , estimator):
2   """Return estimate of energy from estimator
3
4   Parameters:
5     params (ndarray): Array of ansatz parameters
6     ansatz (QuantumCircuit): Parameterized ansatz circuit
7     hamiltonian (SparsePauliOp): Operator representation of
8       Hamiltonian
9     estimator (Estimator): Estimator primitive instance
10
11   Returns:
12     float: Energy estimate
13   """

```

```

14 estimator_job = estimator.run([(ansatz, hamiltonian, [parameters])
15   ])
16 estimator_result = estimator_job.result()[0]
17
18 cost = estimator_result.data.evs[0]
19 return cost
20
21 from qiskit.primitives import StatevectorEstimator
22 estimator = StatevectorEstimator()

```

We can use this cost function to calculate the optimal parameters

```

1 # SciPy minimizer routine
2 from scipy.optimize import minimize
3
4 x0 = np.ones(8)
5
6 result = minimize(cost_func_vqe, x0, args=(ansatz, observable,
7   estimator), method="COBYLA")
8
9 result
10
11 message: Optimization terminated successfully.
12 success: True
13 status: 1
14 fun: -5.99999988971083
15 x: [ 1.779e+00  9.443e-01  1.571e+00  2.105e-05  1.938e+00
16      1.204e+00  6.059e-01  6.059e-01]
17 nfev: 136
18 maxcv: 0.0

```

Step2: Optimize problems for quantum execution We will select the least-busy backend, and import the necessary components from the qiskit_ibm_runtime.

```

1 from qiskit_ibm_runtime import SamplerV2 as Sampler
2 from qiskit_ibm_runtime import EstimatorV2 as Estimator
3 from qiskit_ibm_runtime import Session, Options
4 from qiskit_ibm_runtime import QiskitRuntimeService
5
6
7 service = QiskitRuntimeService(channel='ibm_quantum')
8 backend = service.least_busy(operational=True, simulator=False)

```

We will transpile the circuit using the preset pass manager with optimization level 3, and we will apply the corresponding layout to the observable.

```

1 from qiskit.transpiler.preset_passmanagers import
2 generate_preset_pass_manager

```

```

2 pm = generate_preset_pass_manager(backend=backend, optimization_level
3   =3)
4 isa_ansatz = pm.run(ansatz)
5 isa_observable = observable.apply_layout(layout = isa_ansatz.layout)

```

Step3: Execute using Qiskit Runtime Primitives We are now ready to run our calculation on IBM Quantum hardware. Because the cost function minimization is highly iterative, we will start at a Runtime session. This way we only have to wait in a queue once. Once the job begins running, each iteration with updates to parameters will run immediately.

```

1 #Estimated required time: <20 min. Benchmarked at 17 min, 40 s on
2   ibm-osaka , on 5-29-24
3
4 x0 = np.ones(8)
5
6 with Session(backend=backend) as session:
7     session_options = Options()
8     session_options.execution.shots = 4096
9     session_options.resilience_level = 1
10
11 estimator = Estimator(session=session)
12 sampler = Sampler(session = session)
13 estimator.options.default_shots = 10_000
14
15 result = minimize(cost_func_vqe, x0, args=(isa_ansatz,
16   isa_observable, estimator), method="COBYLA")
17 session.close()
18 print(result)

```

```

1 message: Optimization terminated successfully .
2 success: True
3 status: 1
4 fun: -5.633934465186281
5 x: [ 1.135e+00  1.712e+00  1.340e+00  9.429e-02  2.175e+00
6           7.089e-01  4.387e-01  5.427e-01]
7 nfev: 90
8 maxcv: 0.0

```

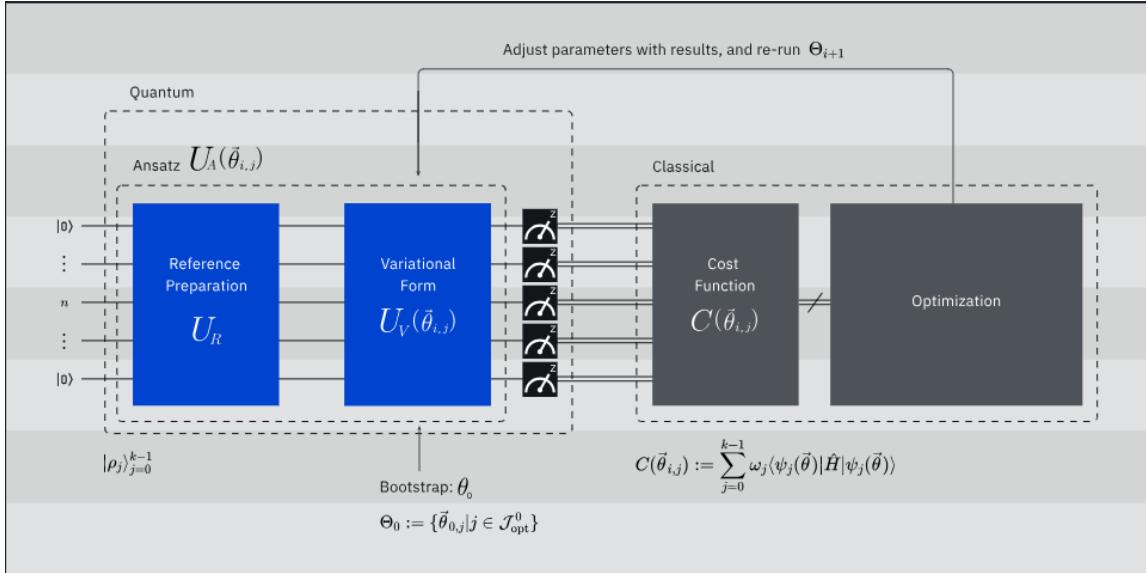
Step4: Post-process, return in classical format We can see that the minimization routine successfully terminated, meaning we reached the default tolerance of the COBYLA classical optimizer. If we require a more precise result, we can specify a smaller tolerance. This may indeed be the case, since the result was several percent off compared to the result obtained by the simulator above.

The value of x obtained is the current best guess for the parameters that minimize the cost function. If iterating to obtain a higher precision, those values should be used in place of the x_0 initially used (a vector of ones).

Finally, we note that the function was evaluated 96 times in the process of optimization. That might be different from the number of optimization steps, since some optimizers require multiple function evaluations in a single step, such as when estimating a gradient.

23.6.2 Subspace Search VQE (SSVQE)

SSVQE is a variant of VQE that allows obtaining the first k eigenvalues of an observable \hat{H} with eigenvalues $\{\lambda_0, \lambda_1, \dots, \lambda_{N-1}\}$, where $N \geq k$. Without loss of generality, we assume that $\lambda_0 < \lambda_1 < \dots < \lambda_{N-1}$. SSVQE introduces a new idea by adding weights to help prioritize optimizing for the term with the largest weight. To



implement this algorithm, we need k mutually orthogonal reference states $\{|\rho_j\rangle\}_{j=0}^{k-1}$, meaning $\langle \rho_j | \rho_l \rangle = \delta_{jl}$ for $j, l < k$. These states can be constructed using Pauli

operators. The cost function of this algorithm is then:

$$\begin{aligned} C(\vec{\theta}) &= \sum_{j=0}^{k-1} w_j \langle \rho_j | U_V^\dagger(\vec{\theta}) \hat{H} U_V(\vec{\theta}) | \rho_j \rangle \\ &= \sum_{j=0}^{k-1} w_j \langle \psi_j(\vec{\theta}) | \hat{H} | \psi_j(\vec{\theta}) \rangle \end{aligned}$$

where w_j is an arbitrary positive number such that if $j < l < k$ then $w_j > w_l$, and $U_V(\vec{\theta})$ is the user-defined variational form.

The SSVQE algorithm relies on the fact that eigenstates corresponding to different eigenvalues are mutually orthogonal. Specifically, the inner product of $U_V(\vec{\theta}) |\rho_j\rangle$ and $U_V(\vec{\theta}) |\rho_l\rangle$ can be expressed as:

$$\begin{aligned} \langle \rho_j | U_V^\dagger(\vec{\theta}) U_V(\vec{\theta}) | \rho_l \rangle &= \langle \rho_j | I | \rho_l \rangle \\ &= \langle \rho_j | \rho_l \rangle \\ &= \delta_{jl} \end{aligned}$$

The first equality holds because $U_V(\vec{\theta})$ is a quantum operator and is therefore unitary. The last equality holds because of the orthogonality of the reference states $|\rho_j\rangle$. The fact that orthogonality is preserved through unitary transformation is deeply related to the principle of conservation of information, as expressed in quantum information science. Under the view, non-unitary transformations represent processes where information is either lost or injected.

Weights w_j help ensure that all the states are eigenstates. If the weights are sufficiently different, the term with the largest weight (*i.e.*, w_0) will be given priority during optimization over the others. As a result, the resulting state $U_V(\vec{\theta}) |\rho_0\rangle$ will become the eigenstate corresponding to λ_0 . Because $\{U_V(\vec{\theta}) |\rho_j\rangle\}_{j=0}^{k-1}$ are mutually orthogonal, the remaining states will be orthogonal to it and, therefore, contained in the subspace corresponding to the eigenvalues $\{\lambda_1, \dots, \lambda_{N-1}\}$.

Applying the same argument to the rest of the terms, the next priority would then be the term with weight w_1 , so $U_V(\vec{\theta}) |\rho_1\rangle$ would be the eigenstate corresponding to λ_1 , and the other terms would be contained in the eigenspace of $\{\lambda_2, \dots, \lambda_{N-1}\}$.

By reasoning inductively, we deduce that $U_V(\vec{\theta}) |\rho_j\rangle$ will be an approximate eigenstates of λ_j for $0 \leq j < k$.

SSVQE's Theoretical layout

SSVQE's can be summarized as follows:

- Prepare several reference states by applying a unitary U_R to k different computational basis states
- This algorithm requires the usage of k mutually orthogonal reference states $\{|\rho_j\rangle\}_{j=0}^{k-1}$ such that $\langle \rho_j | \rho_l \rangle = \delta_{jl}$ for $j, l < k$.
- Apply the variational form $U_V(\vec{\theta}_{i,j})$ to each reference state ,resulting in the following ansatz $U_A(\vec{\theta}_{i,j})$.
- Bootstrap at $i = 0$ if a similar problem is available (usually found via classical simulation or sampling).
- Evaluate the cost function $C(\vec{\theta}_{i,j}) = \sum_{j=0}^{k-1} w_j \langle \psi_j(\vec{\theta}) | \hat{H} | \psi_j(\vec{\theta}) \rangle$ for all prepared states on a quantum computer.
- This can be separated into calculating the expectation value for an observable $\langle \psi_j(\vec{\theta}) | \hat{H} | \psi_j(\vec{\theta}) \rangle$ and multiplying the result by w_j .
- Afterward, the cost function returns the sum of all weighed expectation values.
- Use a classical optimizer to determine the next set of parameters Θ_{i+1} .
- Repeat the above steps until convergence is achieved.

You will be reconstructing SSVQE's cost function in the assessment, but we have the following snippet to motivate your solution:

```

1 import numpy as np
2
3 def cost_func_ssvqe(params, initialized_anastz_list, weights, ansatz,
4     hamiltonian, estimator):
5     """Return estimate of energy from estimator
6
7     # Parameters:
8     #     params (ndarray): Array of ansatz parameters
9     #     initialized_anastz_list (list QuantumCircuit): Array of
10    #         initialised ansatz with reference
11    #     weights (list): List of weights
12    #     ansatz (QuantumCircuit): Parameterized ansatz circuit
13    #     hamiltonian (SparsePauliOp): Operator representation of
14    #         Hamiltonian
15    #     estimator (Estimator): Estimator primitive instance
16
17    # Returns:
18    #     float: Weighted energy estimate

```

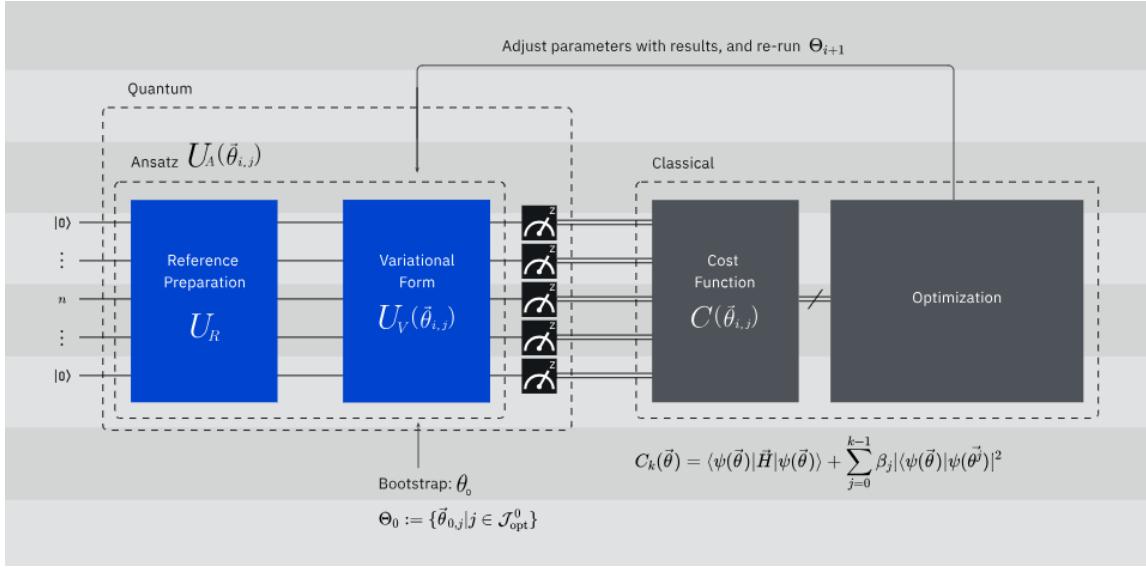
```

16 # """
17
18 energies = []
19
20 # Define SSVQE
21
22 weighted_energy_sum = np.dot(energies, weights)
23 return weighted_energy_sum

```

23.6.3 Variational Quantum Deflation

VQD is an alternative method that extends VQE to obtain the first k -eigenvalues of an observable \hat{H} with eigenvalues $\{\lambda_0, \lambda_1, \dots, \lambda_{N-1}\}$, where $N \geq k$, instead of only the first. For the rest of this section, we will assume, without loss of generality, that $\lambda_0 \leq \lambda_1 \leq \dots \leq \lambda_{N-1}$. VQD introduces the notion of a penalty cost to guide the optimization process. VQD introduces a penalty term, introduced as β , to balance



the contribution of each overlap term to the cost. This penalty term serves to penalize the optimization process if orthogonality is not achieved. We impose this constraint because the eigen states of an observable, or a Hermitian operator, corresponding to different eigenvalues are mutually orthogonal, or can be made so in the case of degeneracy or repeated eigenvalues. Thus, by enforcing orthogonality with the eigenstate corresponding to λ_0 , we are effectively optimizing over the subspace that corresponds to the rest of the eigenvalues $\{\lambda_1, \lambda_2, \dots, \lambda_{N-1}\}$. here λ_1 is the lowest

eigenvalue from the rest of the eigenvalues and, therefore, the optimal solution to the new problem can be obtained using the variational theorem.

The general idea behind VQE is to use VQE as usual to obtain the lowest eigenvalue $\lambda_0 = C_0(\vec{\theta}^0) = C_{VQE}(\vec{\theta}^0)$ along with the corresponding (approximate) eigenstate $|\psi(\vec{\theta}^0)\rangle$ for some optimal parameter vector $\vec{\theta}^0$. Then, to obtain the next eigenvalue $\lambda_1 > \lambda_0$, instead of minimizing the cost function $C_0(\vec{\theta}) = \langle\psi(\vec{\theta})|\hat{H}|\psi(\vec{\theta})\rangle$, we optimize:

$$C_1(\vec{\theta}) = C_0(\vec{\theta}) + \beta_0 |\langle\psi(\vec{\theta})|\psi(\vec{\theta}^0)\rangle|^2$$

The positive value β_0 should be greater than $\lambda_1 - \lambda_0$.

This introduces a new cost function that can be viewed as a constrained problem, where we minimize $C_{VQE}(\vec{\theta}) = \langle\psi(\vec{\theta})|\hat{H}|\psi(\vec{\theta})\rangle$ subject to the constraint that the state must be orthogonal to the previously obtained $|\psi(\vec{\theta}^0)\rangle$, with β_0 acting as a penalty terms if the constraint is not satisfied.

Alternatively, this new problem can be interpreted as running *VQE* on the new observable:

$$\hat{H}_1 = \hat{H} + \beta_0 |\psi(\vec{\theta}^0)\rangle\langle\psi(\vec{\theta}^0)| \implies C_1(\vec{\theta}) = \langle\psi(\vec{\theta})|\hat{H}_1|\psi(\vec{\theta})\rangle,$$

Assuming the solution to the new problem is $|\psi(\vec{\theta}^1)\rangle$, the expected value of \hat{H} (not \hat{H}_1) should be $\langle\psi(\vec{\theta}^1)|\hat{H}|\psi(\vec{\theta}^1)\rangle = \lambda_1$.

To obtain the third eigenvalue λ_2 , the cost function to optimize is:

$$C_2(\vec{\theta}) = C_1(\vec{\theta}) + \beta_1 |\langle\psi(\vec{\theta})|\psi(\vec{\theta}^1)\rangle|^2$$

where β_1 is a positive constant large enough to enforce orthogonality of the solution state to both $|\psi(\vec{\theta}^0)\rangle$ and $|\psi(\vec{\theta}^1)\rangle$. This penalizes states in the search space that do not meet this requirement, effectively restricting the search space. Thus, the optimal solution of the new problem should be the eigenstate corresponding to λ_2 .

Like the previous case, this new problem can also be interpreted as VQE with the observable:

$$\hat{H}_2 = \hat{H}_1 + \beta_1 |\psi(\vec{\theta}^1)\rangle\langle\psi(\vec{\theta}^1)| \implies C_2(\vec{\theta}) = \langle\psi(\vec{\theta})|\hat{H}_2|\psi(\vec{\theta})\rangle$$

If the solution to this new problem is $|\psi(\vec{\theta}^2)\rangle$, the expected value of \hat{H} (not \hat{H}_2) should be $\langle\psi(|theta|^2)|\hat{H}|\psi(\vec{\theta}^2)\rangle = \lambda_2$.

Analogously, to obtain the k th eigenvalue λ_{k-1} , you would minimize the cost function:

$$C_{k-1}(\vec{\theta}) = C_{k-2}(\vec{\theta}) + \beta_{k-2} |\langle\psi(\vec{\theta})|\psi(\vec{\theta}^{k-2})\rangle|^2,$$

Remember that we defined $\vec{\theta}^j$ such that $\langle \psi(\vec{\theta}^j) | \hat{H} | \psi(\vec{\theta}^j) \rangle = \lambda_j, \forall j < k$. This problem is equivalent to minimizing $V(\vec{\theta}) = \langle \psi(\vec{\theta}) | \hat{H} | \psi(\vec{\theta}) \rangle$ but with the constraining that the state must be orthogonal to $|\psi(\vec{\theta}^j)\rangle; \forall j \in 0, \dots, k-1$, thereby restricting the search space to the subspace corresponding to the eigenvalues $\{\lambda_{k-1}, \dots, \lambda_{N-1}\}$.

This problem is equivalent to a VQE with the observable:

$$\hat{H}_{k-1} = \hat{H}_{k-2} + \beta_{k-2} |\psi(\vec{\theta}^{k-2})\rangle \langle \psi(\vec{\theta}^{k-2})| \implies C_{k-1}(\vec{\theta}) = \langle \psi(\vec{\theta}) | \hat{H}_{k-1} | \psi(\vec{\theta}) \rangle,$$

As you can see from the process, to obtain the k -th eigenvalue, you need the (approximate) eignestates of the previous $k-1$ eigenvalues, so you would need to run VQE a total of k times. Therefore, VQD's cost function is as follows:

$$C_k(\vec{\theta}) = \langle \psi(\vec{\theta}) | \hat{H} | \psi(\vec{\theta}) \rangle + \sum_{j=0}^{k-1} \beta_j |\langle \psi(\vec{\theta}) | \psi(\vec{\theta}^j) \rangle|^2$$

VQD's Theoretical Layout

VQD's layout can be summarized as follows:

- Prepare a reference operator U_R .
- Apply a variational form $U_V(\vec{\theta}_{i,j})$ to the reference state, creating the following ansatze $U_A(\vec{\theta}_{i,j})$.
- Bootstrap at $i = 0$ if we have a similar problem (typically found via classical simulation or sampling)
- Evaluate the cost function $C_k(\vec{\theta})$ which involved computing k excited states and an array of β 's defining the overall penalty for each overlap term.
- Calculate the expectation value for an observable $\langle \psi_j(\vec{\theta}) | \hat{H} | \psi_j(\vec{\theta}) \rangle$ for each k
- Calculate the penalty $\sum_{j=0}^{k-1} \beta_j |\langle \psi(\vec{\theta}) | \psi_j(\vec{\theta}) \rangle|^2$.
- The cost function should then return the sum of these two terms.
- Use a classical optimizer to choose the next set of parameters Θ_{i+1}
- Repeat this process until convergence is reached.

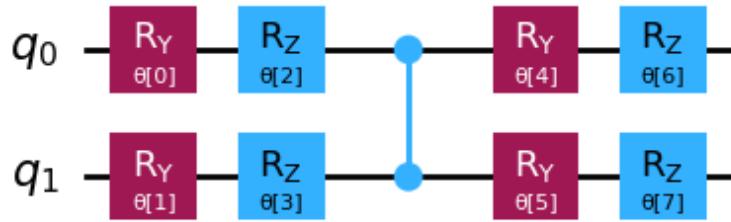
VQD's Implementation

For this implementation, we'll create a function for an overlap penalty. This penalty will be used in the cost function at each iteration. This process will be repeated for each excited state

```

1 from qiskit.circuit.library import TwoLocal
2
3 raw_ansatz = TwoLocal(2, rotation_blocks=["ry", "rz"],
4                       entanglement_blocks="cz", reps=1)
5 raw_ansatz.decompose().draw('mpl')

```



First, we'll setup a fuction that calculates the sstate fidelity - a percentage of overlap between two states that we'll use as a penalty for VQD.

```

1 import numpy as np
2
3 def calculate_overlaps(ansatz, prev_circuits, parameters, sampler):
4
5     def create_fidelity_circuit(circuit_1, circuit_2):
6
7         """
8             Constructs the list of fidelity circuits to be evaluated.
9             These circuits represent the state overlap between pairs of
10            input circuits,
11            and their construction depends on the fidelity method
12            implementations.
13            """
14
15
16        if len(circuit_1.clbits) > 0:
17            circuit_1.remove_final_measurements()
18        if len(circuit_2.clbits) > 0:
19            circuit_2.remove_final_measurements()

```

```

17     circuit = circuit_1.compose(circuit_2.inverse())
18     circuit.measure_all()
19     return circuit
20
21 overlaps = []
22
23 for prev_circuit in prev_circuits:
24     fidelity_circuit = create_fidelity_circuit(ansatz, prev_circuit)
25     sampler_job = sampler.run([(fidelity_circuit, parameters)])
26     meas_data = sampler_job.result()[0].data.meas
27
28     counts_0 = meas_data.get_int_counts().get(0, 0)
29     shots = meas_data.num_shots
30     overlap = counts_0 / shots
31     overlaps.append(overlap)
32
33 return np.array(overlaps)

```

It's time to write VQD's cost function. As before wen we calculated only the ground state, we will determine the lowest energy state using the Estimator primitive. However, as described above, we will now add a penalty term to ensure orthogonality of higher-energy states. That is, for each excited state, a penalty is added for any overlap between the current variational state and the lower-energy eigenstate already found.

```

1 def cost_func_vqd(parameters, ansatz, prev_states, step, betas,
2 estimator, sampler, hamiltonian):
3
4     estimator_job = estimator.run([(ansatz, hamiltonian, [parameters])])
5
6     total_cost = 0
7
8     if step > 1:
9         overlaps = calculate_overlaps(ansatz, prev_states, parameters,
10                                     sampler)
11         total_cost = np.sum([np.real(betas[state] * overlap) for state,
12                             overlap in enumerate(overlaps)])
13
14     estimator_result = estimator_job.result()[0]
15
16     value = estimator_result.data.evs[0] + total_cost
17
18     return value

```

Note specially that the cost function above refers to the calculate_overlaps function, which actually creates a new quantum circuit. If we want to run on a real hardware, the new circuit must also be transpiled, hopefully in an optimal way, to run on the backend we select. Note that transpilation has been built to the calculate_overlaps or cost_func_vqd functions. Feel free to try modifying the code yourself to build in this additional (and conditional) transpilation - but this will also be done for you in the next lesson.

In this lesson, we will run the VQD algorithm using the Statevector Sampler and Statevector Estimator:

```

1 from qiskit.primitives import StatevectorSampler as Sampler
2 from qiskit.primitives import StatevectorEstimator as Estimator
3 sampler = Sampler()
4 estimator = Estimator()
```

We will introduce an observable to be estimated. In the next lesson we will add some physical context to this, like the excited state of a molecule. It may be helpful to think of this observable as the Hamiltonian of a system that can have excited states, even though this observable has not been chosen to match any particular molecule or atom.

```

1 from qiskit.quantum_info import SparsePauliOp
2 observable = SparsePauliOp.from_list([('II', 2), ('XX', -2), ('YY', 3),
                                         ('ZZ', -3)])
```

Here, we set the total number of states we wish to calculate (ground state and excited states, k), and the penalties (betas) for overlap between statevectors that should be orthogonal. The consequences of choosing betas to be too high or too low will be explored a bit in the next lesson. For now, we will simply use those provided below. We will start by using all zeros as our parameters. We will start by using all zeros as our parameters. In your own calculations, you may want to use more clever starting parameters based on your knowledge of the system or on previous calculations.

```

1 k = 3
2 betas = [33, 33, 33]
3 x0 = np.zeros(8)
```

We can now run the calculation:

```

1 from scipy.optimize import minimize
2
3 prev_states = []
4 prev_opt_parameters = []
5 eigenvalues = []
6
```

```

7 for step in range(1, k + 1):
8
9     if step > 1:
10         prev_states.append(ansatz.assign_parameters(prev_opt_parameters
11 ))
12
13     result = minimize(cost_func_vqd, x0, args=(ansatz, prev_states,
14     step, betas, estimator, sampler, observable), method="COBYLA",
15     options={'maxiter': 200,})
16     print(result)
17
18     prev_opt_parameters = result.x
19     eigenvalues.append(result.fun)

message: Optimization terminated successfully.
success: True
status: 1
fun: -5.99999983468767
x: [-5.092e-01 -5.806e-02 -1.571e+00 -1.722e-05 2.688e-01
     -2.688e-01 -8.556e-01 -8.555e-01]
nfev: 125
maxcv: 0.0
message: Optimization terminated successfully.
success: True
status: 1
fun: 4.025471915908117
x: [ 1.719e-01 -3.279e-01 1.567e+00 -5.447e-02 5.202e-03
     5.401e-02 3.745e-02 -3.008e-01]
nfev: 99
maxcv: 0.0
message: Optimization terminated successfully.
success: True
status: 1
fun: 5.923215165670577
x: [ 1.464e+00 -7.677e-01 -1.584e+00 -6.150e-02 6.260e-01
     1.752e+00 3.236e+00 -2.544e-01]
nfev: 86
maxcv: 0.0

```

This values we obtained from the cost function are approximately $-6.00, 4.02, 5.92$. The important thing about these results is that the function values are increasing. If we have obtained a first excited state that is lower in energy than our initial, unconstrained calculation of the ground state, that would have indicated an error somewhere in our code.

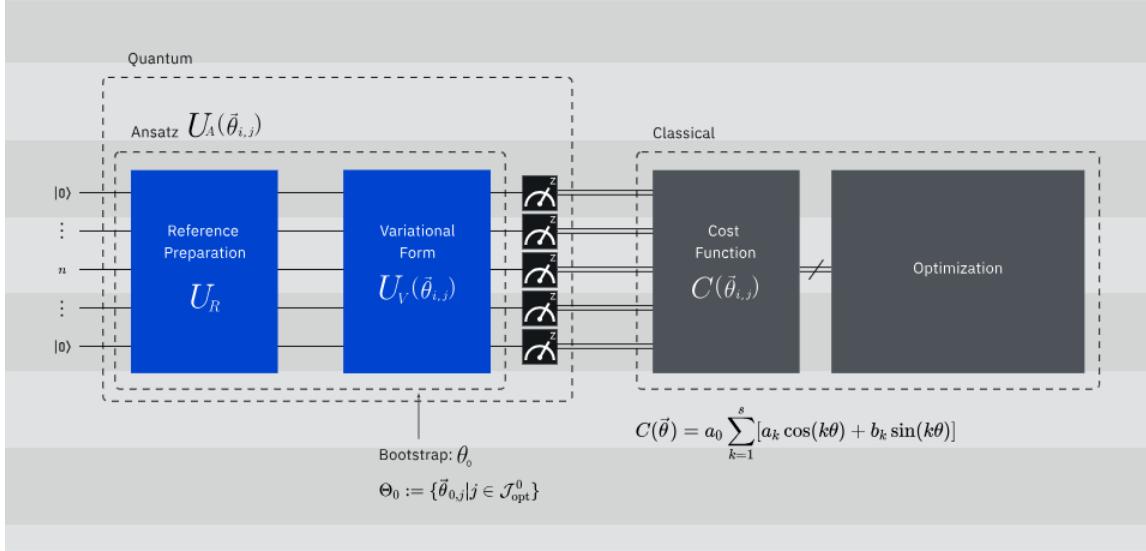
The values of x are the parameters that yielded a state vector corresponding to

each of these costs/energies.

Finally, we note that all three minimization converged to within the default tolerance of the classical optimizers (here COBYLA). They required 125, 99, 86 function evaluations, respectively.

23.6.4 Quantum Sample Regression (QSR)

One of the main issues with VQE is the multiple calls to a quantum computer that are required to obtain the parameters for each step, including k , $k-$, etc. This is especially problematic when access to quantum devices is queued. While a Session can be used to group multiple iterative calls, an alternative approach is to use sampling. By utilizing more classical resources, we can complete the full optimization process in a single call. This is where, Quantum Sampling Regressions comes into play. Single access of quantum computers is still a low-offer/high-demand commodity, we find this trade-off to be both possible and convenient for many current studies. This approach harnesses all available classical capabilities while still capturing many of the inner workings and intrinsic properties of quantum computations that do not appear in simulation. The idea behind QSR is that the cost function $C(\theta) = \langle \psi(\theta) | \hat{H} | \psi(\theta) \rangle$



can be expressed as a Fourier series in the following manner:

$$\begin{aligned} C(\vec{\theta}) &= \langle \psi(\theta) | \hat{H} | \psi(\theta) \rangle \\ &= a_0 + \sum_{k=1}^S [a_k \cos(k\theta) + b_k \sin(k\theta)] \end{aligned}$$

Depending on the periodicity and bandwidth of the original function, the set S may be finite or infinite. For the purposes of the discussion, we will assume it is infinite. The next step is to sample the cost function $C(\theta)$ multiple times in order to obtain the Fourier coefficients $\{a_0, a_k, b_k\}_{k=1}^S$. Specifically, since we have $2S + 1$ unknowns, we will need to sample the cost function $2S + 1$ times.

If we then sample the cost function for $2S + 1$ parameter values $\{\theta_1, \dots, \theta_{2S+1}\}$, we can obtain the following system:

$$\begin{pmatrix} 1 & \cos(\theta_1) & \sin(\theta_1) & \cos(2\theta_1) & \dots & \sin(S\theta_1) \\ 1 & \cos(\theta_2) & \sin(\theta_2) & \cos(2\theta_2) & \dots & \sin(S\theta_2) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \cos(\theta_{2S+1}) & \sin(\theta_{2S+1}) & \cos(2\theta_{2S+1}) & \dots & \sin(S\theta_{2S+1}) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ a_2 \\ \vdots \\ b_s \end{pmatrix} = \begin{pmatrix} C(\theta_1) \\ C(\theta_2) \\ \vdots \\ C(\theta_{2S+1}) \end{pmatrix}$$

that we'll rewrite as

$$Fa = c$$

In practice, this system is generally not consistent because the cost function values c are not exact. Therefore, it is usually a good idea to normalize them by multiplying them by F^\dagger on the left, which results in:

$$F^\dagger Fa = F^\dagger c$$

This new system is always consistent, and its solution is a least-squares solution to the original problem. If we have k parameters instead of just one, and each parameter θ^i has its own S_i for $i \in 1, \dots, k$, then the total number of samples required is:

$$T = \prod_{i=1}^k (2S_i + 1) \leq \prod_{i=1}^k (2S_{max} + 1) = (2S_{max} + 1)^n$$

where $S_{max} = \max_i(S_i)$. Furthermore, adjusting S_{max} as a tunable parameter (instead of inferring it) opens up new possibilities, such as:

- **Oversampling:** to improve accuracy.
- **Undersampling:** to boost performance by reducing runtime overhead or eliminating local minima.

QSR's Theoretical Layout

QSR's layout can be summarized as follows:

- Prepare reference operators U_R .
- We'll go from the state $|0\rangle$ to the reference state $|\rho\rangle$.
- Apply the variational form $U_V(\vec{\theta}_{i,j})$ to create an ansatz $U_A(\vec{\theta}_{i,j})$.
- Determine the bandwidth associated with each parameter in the ansatz. An upper bound is sufficient.
- Bootstrap at $i = 0$ if we have a similar problem (typically found via classical simulation or sampling)
- Sample the cost function $C(\vec{\theta}) = a_0 + \sum_{k=1}^S [a_k \cos(k\theta) + b_k \sin(k\theta)]$ at least T times- $T = \prod_{i=1}^k (2S_i + 1) \leq \prod_{i=1}^k (2S_{max} + 1) = (2S_{max} + 1)^n$
- Decide whether to oversample/undersample to balance speed vs accuracy by adjusting T .
- Compute the Fourier coefficients from the samples (i.e., solve the normalized linear system of equations).
- Solve for the global minimum of the resulting regression function on a classical machine.

23.6.5 Summary

With this lesson, you learned about multiple variational instances available:

- General layout
- Introducing weights and penalties to adjust a cost function.
- Exploring undersampling vs oversampling to trade-off speed vs accuracy.

These ideas can be adapted to form a custom variational algorithm that fits your problem. We encourage you to share your results with the community. The next section we will explore how to use a variational algorithm to solve an application.

23.7 Variational Quantum Eigensolver

Usage Estimate: 73 minutes on ibm_kyiv or 28 minutes on ibm_manila. (NOTE:This is an estimate only. Your runtime may vary.)

23.7.1 Background

Variational quantum algorithms are promising candidate hybrid-quantum algorithms for observing quantum computational utility on noisy near-term devices. Variational algorithms are characterized by the use of classical optimization algorithm to iteratively update a parameterized trial solution, or “ansatz”. Chief among these methods is the Variational Quantum Eigensolver (VQE) that aims to solve for the ground state of a given Hamiltonian represented as a linear combination of Pauli terms, with an ansatz circuit where the number of parameters to optimize over is polynomial in number of qubits. Given that the size of the full solution vector is exponential in the number of qubits, successful minimization using VQE requires, in general, additional problem-specific information to define the structure of the ansatz circuit.

Executing a VQE algorithm requires the following components:

1. Hamiltonian and ansatz (problem specification)
2. Qiskit Runtime estimator
3. Classical Optimizer

Although the Hamiltonian and ansatz require domain specific knowledge to construct, these details are immaterial to the Runtime, and we execute a wide class of VQE problems in the same manner.

23.7.2 Requirements

Before starting this tutorial, ensure that you have the following installed:

- Qiskit SDK 1.0 or later, with visualization support (`pip install 'qiskit[visulaization]'`)
- Qiskit Runtime (`pip install qiskit-ibm-runtime`) 0.22 or later
- SciPy(`python -m pip install scipy`)

23.7.3 Setup

Here we import the tools needed for a VQE experiment.

```

1 # General imports
2 import numpy as np
3
4 # Pre-defined ansatz circuit and operator class for Hamiltonian
5 from qiskit.circuit.library import EfficientSU2
6 from qiskit.quantum_info import SparsePauliOp
7
8 # SciPy minimizer routine
9 from scipy.optimize import minimize
10
11 # Plotting functions
12 import matplotlib.pyplot as plt

```

No output produced

```

1 # runtime imports
2 from qiskit_ibm_runtime import QiskitRuntimeService, Session
3 from qiskit_ibm_runtime import EstimatorV2 as Estimator
4
5 # To run on hardware, select the backend with the fewest number of jobs
      in the queue
6 service = QiskitRuntimeService(channel="ibm_quantum")
7 backend = service.least_busy(operational=True, simulator=False)

```

Step1: Map classical inputs to a quantum problem

Although the problem instance in question for the VQE algorithm can come from a variety of domains, the form for execution through Qiskit Runtime is the same. Qiskit provides a convenience class for expressing hamiltonians in Pauli form, and a collection of widely used ansatz circuits in the `qiskit.circuit.library`.

this example Hamiltonian is derived from a quantum chemistry problem.

```

1 hamiltonian = SparsePauliOp.from_list(
2     [("YZ", 0.3980), ("ZI", -0.3980), ("ZZ", -0.0113), ("XX", 0.1810)]
3 )

```

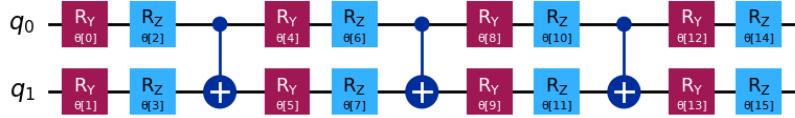
Our choice of ansatz is the `EfficientSU2` that, by default, linearly entangles qubits, making it ideal for quantum hardware with limited connectivity.

```

1 ansatz = EfficientSU2(hamiltonian.num_qubits)
2 ansatz.decompose().draw("mpl", style="iqp")

```

Output:



From the previous figure we see that our ansatz circuit is defined by a vector of parameters, θ_i , with the total number given by:

```
1 num_params = ansatz.num_parameters
2 num_params
```

Output:

```
1 16
```

Step2: Optimize problem for quantum execution

To reduce the total job execution time, Qiskit primitives only accept circuits (ansatz) and observables (Hamiltonian) that conform to the instructions and connectivity supported by the target system (referred to as instruction set architecture (ISA)circuits and observables).

ISA circuit

Schedule a series of qiskit.transpiler passes to optimize the circuit for a selected backend and make it compatible with the backend's ISA. This can be easily done with a preset pass manager from qiskit.transpiler and its optimization_level parameter.

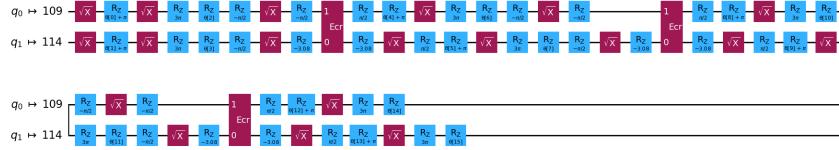
The lowest optimization level does the minimum needed to get the circuit running on the device; it maps the circuit qubits to the device qubits and adds swap gates to allow all two-qubit operations. The highest optimization level is much smarter and uses lots of tricks to reduce the overall gate count. Since multi-qubit gates have high error rates and qubits decohere over time, the shorter circuits should give better results.

```
1 from qiskit.transpiler.preset_passmanagers import
2     generate_preset_pass_manager
3
4 target = backend.target
5 pm = generate_preset_pass_manager(target=target, optimization_level=3)
6 ansatz_isa = pm.run(ansatz)
```

No Output produced

```
1 ansatz_isa.draw(output="mpl", idle_wires=False, style="iqp")
```

Output:



ISA observable

Transform the Hamiltonian to make it backend-compatible before running jobs with Runtime Esimtaor V2. Perform the transformation by using the `apply_layout` method of `SparsePauliOp` object.

```
1 hamiltonian_isa = hamiltonian.apply_layout(layout=ansatz_isa.layout)
```

Step 3: Execute using Qiskit Primitives

Like many classical optimization problems, the solution to a VQE problem can be formulated as a minimization of a scalar cost function. By definition, VQE looks to find the ground state solution to a Hamiltonian by optimizing the ansatz circuit parameters to minimize the expectation value (energy) of the Hamiltonian. With the Qiskit Runtime Estimator directly taking a Hamiltonian and parameterized ansatz, and returning the necessary energy, the cost function for a VQE instance is quite simple.

Note that `run()` method of Qiskit Runtime Estimator V2 takes an iterable of primitive unified blocs (PUBs). Each PUB is an iterable in the format (`circuit`, `observables`, `parameter_values`: Optional, `precision`: Optional)

```
1 def cost_func(params, ansatz, hamiltonian, estimator):
2     """Return estimate of energy from estimator
3
4     Parameters:
5         params (ndarray): Array of ansatz parameters
6         ansatz (QuantumCircuit): Parameterized ansatz circuit
7         hamiltonian (SparsePauliOp): Operator representation of
8             Hamiltonian
9         estimator (EstimatorV2): Estimator primitive instance
10        cost_history_dict: Dictionary for storing intermediate results
```

```

10
11     Returns:
12         float: Energy estimate
13     """
14     pub = (ansatz, [hamiltonian], [params])
15     result = estimator.run(pub).result()
16     energy = result[0].data.evs[0]
17
18     cost_history_dict["iters"] += 1
19     cost_history_dict["prev_vector"] = params
20     cost_history_dict["cost_history"].append(energy)
21     print(f"Iters. done: {cost_history_dict['iters']} [Current cost: {energy}]")
22
23     return energy

```

Note that, in addition to the array of optimization parameters that must be the first argument, we use additional arguments to pass the terms needed in the cost function, such as the `cost_history_dict`. This dictionary stores the current vector at each iteration, for example in case you need to restart the routine due to failure, and also returns the current iteration number and average time per iteration.

```

1 cost_history_dict = {
2     "prev_vector": None,
3     "iters": 0,
4     "cost_history": [],
5 }

```

We can now use a classical optimizer of our choice to minimize the cost function. Here, we use the COBYLA routine from SciPy through the `minimize` function. Note that when running on real quantum hardware, the choice of optimizer is important, as not all optimizers handle noisy cost function landscapes equally well.

To begin the routine, specify a random initial set of parameters:

```

1 x0 = 2 * np.pi * np.random(num_params)

```

```

1 x0

```

Output:

```

1 array([4.41731016, 3.07910816, 2.24090817, 4.49109315, 1.55032543,
2     3.80189756, 1.28634468, 2.38155768, 0.72212296, 3.15156603,
3     5.16945739, 3.10289414, 3.05826307, 0.24272269, 2.06257592,
4     5.23229471])

```

Because we are sending a large number of jobs that we would like to execute together, we use a `Session` to execute all the generated circuits in one block. Here `args` is the standard SciPy way to supply the additional parameters needed by the cost function.

```

1 with Session(backend=backend) as session:
2     estimator = Estimator(session=session)
3     estimator.options.default_shots = 10000
4
5     res = minimize(
6         cost_func,
7         x0,
8         args=(ansatz_isa, hamiltonian_isa, estimator),
9         method="cobyla",
10    )

```

As the terminus of this routing we have a result in the standard SciPy OptimizeResult format. From this we see that it took nfev number of cost function evaluations to obtain the solution vector of parameter angles (x) that, when plugged into the ansatz circuit, yield the approximate ground state solution we were looking for.

```
1 res
```

Output:

```

1 message: Optimization terminated successfully.
2 success: True
3 status: 1
4     fun: -0.634701203143904
5     x: [ 2.581e+00  4.153e-01 ...  1.070e+00  3.123e+00]
6     nfev: 146
7     maxcv: 0.0

```

Step 4: Post-process, return result in classical format

If the procedure terminates correctly, then the prev_vector and iters values in our cost_history_dict dictionary should be equal to the solution vector and total number of function evaluation, respectively. This is easy to verify:

```
1 all(cost_history_dict["prev_vector"] == res.x)
```

Output:

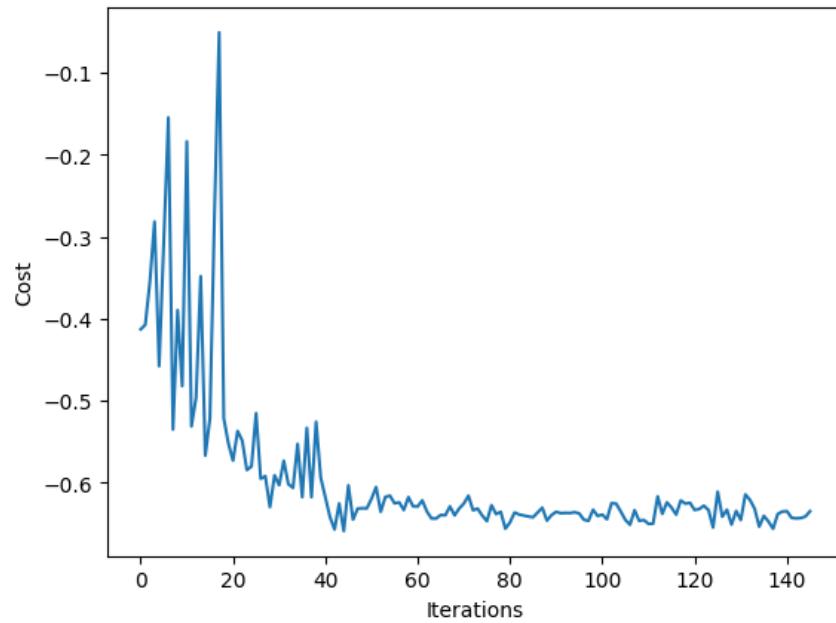
```
1 False
```

We can also now view the progress toward convergence as monitored by the cost history at each iteration:

```

1 fig, ax = plt.subplots()
2 ax.plot(range(cost_history_dict["iters"]), cost_history_dict["cost_history"])
3 ax.set_xlabel("Iterations")
4 ax.set_ylabel("Cost")
5 plt.draw()

```



Output:

```
1 import qiskit_ibm_runtime  
2  
3 qiskit_ibm_runtime.version.get_version_info()
```

Output:

```
1 '0.28.0'
```

```
1 import qiskit  
2  
3 qiskit.version.get_version_info()
```

Output:

```
1 '1.2.0'
```

Part II

Quantum Linear Algebra

At a high level of abstraction, quantum computers compose unitary matrices, and do so with classically unparalleled efficiency. This hints at quantum speedup for linear algebra tasks. However, often one needs to work with large non-unitary matrices thus, for performing general linear algebra tasks we often wish to embed certain non-unitary matrices into unitary matrices represented by efficient quantum circuits, and then apply them to quantum states, and take their sums or products, or implement more general matrix functions. These tasks are collectively referred to as “quantum linear algebra”, the building blocks are discussed in this section.

The techniques described in this section evolved over the past decades and converged to the presented unified framework within several distinct research threads. Block-encodings emerged as a natural approach for embedding non-unitary matrices in quantum circuits, inspired by approaches based on purification, dilation (that is, representing an incoherent state or operation as a coherent one with the help of an ancillary system for example, Stinespring representation or Stinespring dilation), and post-selection. Quantum signal processing (QSP) was discovered as a by product of the characterization of single single-qubit pulse sequences used in nuclear magnetic resonance, for synthesizing polynomial transformations applicable to a “signal parameter” encoded as a matrix element of a single-qubit rotation matrix. Meanwhile, it was extensively studied how matrix functions could be synthesized using the linear combination of unitaries techniques based on matrix exponentials implemented by Hamiltonian simulation, or Chebyshev polynomials of operators implemented via quantum walk techniques. Such matrix exponentials or Chebyshev polynomials can be implemented, e.g, via qubitization of a block-encoded operator. In parallel to progress on advanced amplitude amplification techniques, it was recognized that QSP can be “lifted” for applying polynomial transformation to the eigenvalues of the quantum walk operators (such as those implemented by Qubitization), and thus for implementing a rich family of matrix functions, immediately yielding an optimal algorithm for time independent Hamiltonian simulation. The concepts of qubitization and QSP were later generalized and unified into the framework of quantum singular value transformation, providing generalizations and more efficient implementations of a number of existing quantum algorithms and leading to the discovery of several new algorithms.

Chapter 24

Block Encoding

24.1 Motivation

In this section we describe a generic toolbox for implementing matrix calculations on a quantum computer in an operational way, representing the vectors as quantum states. The matrix arithmetic methodology proposed carries out all calculations in an operational way , such that the matrices are represented by blocks of unitary operators of the quantum system, thereby can in principle result in exponential speedups-in terms of the dimension of the matrices.

Here the results are in an intuitively structured way. First we define how to represent arbitrary matrices as blocks of unitaries, and show how to efficiently encode various matrices this way.

In quantum algorithms, the quantum gates that are applied to quantum states are necessarily unitary operations. However, one often needs to apply a linear transformation to some encoded data that is not represented by a unitary operator, and furthermore one generally needs coherent access to these non-unitary transformations. how can we encode such a non-unitary transformation within a unitary operator? Block-encoding works by embedding the desired linear operator as a suitably normalized block within a larger unitary matrix, such that the full encoding is a unitary operator, and the desired linear operator is given by restricting the unitary to an easily recognizable subspace. To be useful for quantum algorithms, this block encoding must also be realized by some specific quantum circuit acting on the main register and additional ancilla qubits.

Block-encodings are ubiquitous within quantum algorithms, they have both benefits and drawbacks. They are easy to work with, since one can efficiently perform manipulations of block encodings, such as taking products or convex combinations.

On the other hand, this improved working efficiency comes at the cost of having more limited access. For example, if a matrix is stored in classical random access memory, the matrix entries can be explicitly accessed with a single query to the memory, whereas if one only has access to a block-encoding of the matrix, estimating a matrix entry to precision ϵ requires $\mathcal{O}(1/\epsilon)$ uses of the block-encoding unitary in general (by using amplitude estimation subroutine).

Block-encodings also provide a layer of abstraction that assists in the design and analysis of quantum algorithms. One can simply assume access to a block-encoding and count the number of times it is applied. To run, the algorithm, it is necessary to choose a method for implementing the block-encoding. There are many ways of constructing block-encodings that could be suited to the structure of the input. For instance, there are efficient block-encoding strategies for density matrices, positive operator-values measures (POVMs), Gram matrices, sparse-access matrices, matrices that are stored in quantum data structures, structured matrices, and operators given as a linear combination of unitaries (with a known implementation). We discuss these constructions below. For unstructured, dense matrices, the strategy from Gram matrices can be instantiated using state-preparation and quantum random access memory (QRAM) as subroutines.

24.2 Introduction

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).

Many standard Linear algebra problems can be solved on a quantum computer using recently developed quantum linear algebra problems that make use of block encodings and quantum eigenvalue/ singular value transformations.

A block encoding embeds a properly scaled matrix of interest A in a larger Unitary transformation U that can be decomposed into a product of simpler unitaries and implemented efficiently on a quantum computer. Although, quantum algorithms can potentially achieve exponential speedup in solving linear algebra problems compared to the best classical algorithm, such gain in efficiency ultimately hinges on our ability to construct an efficient quantum circuit for the block encoding of A , which is difficult in general, and not trivial even for well structured sparse matrices. In this chapter, we talk about how efficient quantum circuits can be explicitly construct for some well

structured sparse matrices and discuss a few examples and strategies used in these constructions.

In recent years, a new class of quantum algorithms have been developed to solve standard linear algebra problems on quantum computers. These algorithms use the technique of block encoding, to embed a properly scaled matrix A of interest in a larger unitary matrix U_A to a carefully prepared initial state and performing measurements on a subset of qubits. furthermore, if A is Hermitian, by using the technique of the quantum eigenvalue transformation, one can block encode a certain matrix polynomial $p(A)$ by another unitary $U_{p(A)}$ efficiently using U_A as the building block. This procedure can be generalized to non-Hermitian matrices A using a technique called singular value transformation. The larger unitary $U_{p(A)}$ can be decomposed onto a product of simpler unitaries consisting of 2×2 single qubit unitaries, multi-qubit controlled-NOTs, U_A and its Hermitian conjugate. Because approximate solutions to many large-scale linear algebra problems such as linear system of equations, least squares problems and eigenvalue problems (produced by iterative methods) can often be expressed as $p(A)v_0$ for some initial vector v_0 for some initial vector v_0 , the possibility to block encode $p(A)$ will enable us to solve these problems on a quantum computer that perform unitary transformations.

In order to implement this quantum linear algebra algorithms, we need to further express the block encoding matrix U_A as a product of simpler unitaries i.e. we need to express U_A as an efficient quantum circuit. Although it is suggested that such a quantum circuit can be constructed in theory for certain sparse matrices, the proposed construction relies on the availability of “oracle” that can efficiently block encode both the nonzero structure of A and numerical values of the nonzero matrix elements. However, for a general sparse matrix A , finding these “oracles” is entirely non-trivial. Even for sparse matrices that have a well defined non-zero structure and a small set of nonzero matrix elements, encoding the structure and matrix elements by an efficient quantum circuit is not an easy task.

In this chapter, we provide a few accessible examples on how to explicitly construct efficient quantum circuits for some well structured sparse matrices. In particular, some general strategies for writing down the oracles for the non-zero structure and non-zero matrix elements of those matrices. In addition we will also provide circuit diagrams and how these circuits can be easily constructed using Python Qiskit.

The block encoding of A is not unique. We will show a few different encoding schemes and the corresponding quantum circuits. For a sparse matrix A that has at most s nonzero elements per column, which we refer to as an s -sparse matrix, the general strategies we use to construct a quantum circuit actually block encodes A/s . For many applications, the extra scaling factor does not introduce significant issues.

24.3 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}/α 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, j) 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.

24.4 Block Encoding

Block encoding is a technique for embedding a properly scaled nonunitary matrix $A \in \mathbb{C}^{N \times N}$ into a unitary matrix U_A . Note that a properly scaled A , we mean that A is scaled to satisfy $\|A\|_2 = \sigma_{max} \leq 1$. Without such a scaling, a block encoding of A may not exist because the singular values of any submatrix blocks of a unitary matrix must be bounded by 1. Furthermore, if there are some constraints on the type of unitary it can construct e.g. the type of quantum gates available on a quantum computer, we may not be able to find U_A that block encodes A exactly. In this chapter, we assume that A has already been properly scaled, and there is no constraints on the quantum gates we can use to construct a quantum circuit representation of the unitary matrix U_A . 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 (yet to be determined), 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

$$U_A |0, b\rangle = \begin{bmatrix} Ab \\ * \end{bmatrix} = |0\rangle A |b\rangle + |1\rangle |*\rangle$$

Here the unnormalized state $|\perp\rangle = |1\rangle |*\rangle$ 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 24.1. Note that the output state is normalized after the measurement

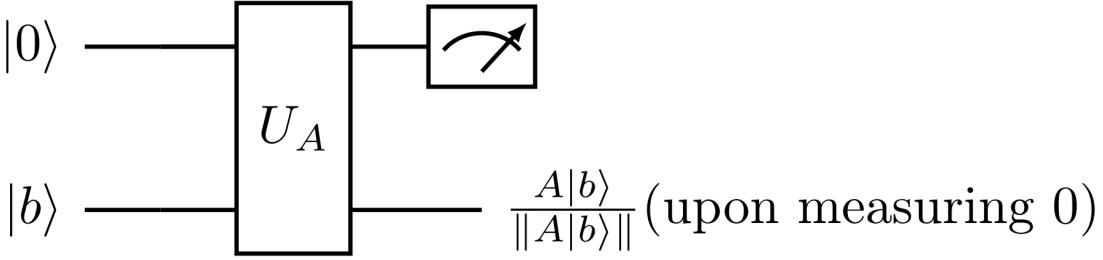


Figure 24.1: Circuit for Block Encoding A using one Ancilla qubit

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

$$U_A = \begin{bmatrix} A & * & \dots & * \\ * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & \dots & * \end{bmatrix}$$

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,

$$(\langle 0^m | \otimes I_n) U_A (|0^m\rangle \otimes I_n) = [I_n \ 0 \ 0 \ \dots \ 0] U_A \begin{bmatrix} I_n \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = A$$

The above expression can be thought of as simply picking the first n columns of U_A and then the first n rows of 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 α 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 α 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 α is chosen to be too large, the probability of obtaining all 0s from the measurement can be vanishingly small.

Our goal is to build a Unitary operator that gives coherent access to an $M \times M$ matrix A (we will later relax the assumption that A is square), with normalization $\alpha \geq \|A\|$, where $\|A\|$ denotes the spectral norm of A . As the name suggests, block encoding is a way of encoding the matrix A as a block in a larger unitary matrix:

$$U_A = \begin{bmatrix} A/\alpha & * \\ * & * \end{bmatrix}$$

More precisely, we say that the unitary U_A is an (α, a, ϵ) -block-encoding of the matrix $A \in \mathbb{C}^{M \times M}$ if

$$\|A - \alpha(\langle 0 |^{\otimes a} \otimes I)U_A(|0\rangle^{\otimes a} \otimes I)\| \leq \epsilon$$

where $a \in \mathbb{N}$ is the number of ancilla qubits used for embedding the block-encoded operator, and $\alpha, \epsilon \in \mathbb{R}_+$ define the normalization and error, respectively. Note that $\alpha \geq \|A\| - \epsilon$ is necessary for U_A to be unitary. The definition above can be extended for general matrices, though additional embedding or padding may be needed (e.g., to make the matrix square).

Once a block-encoding is constructed, it can be used in a quantum algorithm to apply the matrix A to a quantum state by applying the unitary U_A to the larger quantum system. The application of the block-encoding can be thought of as a probabilistic application of A : applying U_A to $|0\rangle^{\otimes a}|\psi\rangle$ and post-selecting on the first register being in the state $|0\rangle^{\otimes a}$ gives an output state proportional to $A|\psi\rangle$ in the second register.

Example 24.4.1. To give an example of block encoding, let us consider a 1×1 matrix $A = \alpha$, where $0 < \alpha < 1$. In this extremely simple case, a block encoding of A can be constructed as

$$U_A = \begin{bmatrix} \alpha & \sqrt{1-\alpha^2} \\ \sqrt{1-\alpha^2} & -\alpha \end{bmatrix}$$

or

$$U_A = \begin{bmatrix} \alpha & -\sqrt{1-\alpha^2} \\ \sqrt{1-\alpha^2} & \alpha \end{bmatrix}$$

Although this type of block encoding can be extended to a properly scaled matrix A of a larger dimension to yield

$$U_A = \begin{bmatrix} A & (I - A^\dagger A)^{1/2} \\ (I - A^\dagger A)^{1/2} & -A \end{bmatrix}$$

or

$$\begin{bmatrix} A & -(I - A^\dagger A)^{1/2} \\ (I - A^\dagger A)^{1/2} & A \end{bmatrix}$$

this approach is not practical because it requires computing the square root of $A^\dagger A$, which requires computing and diagonalizing $A^\dagger A$. In general, there is no efficient algorithm to perform these operations on a quantum computer using $\mathcal{O}(\text{poly}(n))$ quantum gates.

Example 24.4.2. Let H be a Hamiltonian made up of m Pauli terms, meaning that

$$H = \sum_{a=1}^m \lambda_a E_a \text{ where } E_a \text{ is a tensor product of Pauli matrices}$$

Find an algorithm implementing a Unitary close to $e^{-\imath Ht}$, so that $\|U - e^{\imath Ht}\| \leq \epsilon$. Original solutions proceeded by using Trotter approximations: for r large enough,

$$e^{-\imath Ht} \approx (e^{-\imath E_1 t/r} e^{-\imath E_2 t/r} \dots e^{-\imath E_m t/r})^r$$

However, this solution is far from optimal, notably because implementing this approximation, requires $\text{poly}(1/\epsilon)$ gate complexity. Improved algorithms eventually developed into the framework below presented. This framework proceeds by:

1. Defining a type of quantum circuit called a “block-encoding”
2. Showing that, given λ_a and E_a , we can construct an efficient block-encoding of H .
3. Showing that we can get a block-encoding of (an approximation of) $e^{-\imath Ht}$ with few uses of the block-encoding to H .
4. Using this block-encoding to apply our approximation to a state.

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.

Example 24.4.3. A more practical scheme that does not require computing the square root of A can be illustrated by the following real symmetric 2×2 example. Let

$$A = \begin{bmatrix} \alpha_1 & \alpha_2 \\ \alpha_2 & \alpha_1 \end{bmatrix}$$

where $|\alpha_1|, |\alpha_2| \leq 1$. It can be verified that the matrix

$$U_A = \frac{1}{2} \begin{bmatrix} U_\alpha & -U_\beta \\ U_\beta & U_\alpha \end{bmatrix}$$

is a block encoding of $A/2$, where

$$U_\alpha = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_1 & -\alpha_2 \\ \alpha_2 & \alpha_1 & -\alpha_2 & \alpha_1 \\ \alpha_1 & -\alpha_2 & \alpha_1 & \alpha_2 \\ -\alpha_2 & \alpha_1 & \alpha_2 & \alpha_2 \end{bmatrix}$$

and

$$U_\beta = \begin{bmatrix} \beta_1 & \beta_2 & \beta_1 & -\beta_2 \\ \beta_2 & \beta_1 & -\beta_2 & \beta_1 \\ \beta_1 & -\beta_2 & \beta_1 & \beta_2 \\ \beta_1 & -\beta_2 & \beta_1 & \beta_2 \\ -\beta_2 & \beta_1 & \beta_2 & \beta_1 \end{bmatrix}$$

with $\beta_1 = \sqrt{1 - \alpha_1^2}$ and $\beta_2 = \sqrt{1 - \alpha_2^2}$. To implement this unitary on a quantum computer, we must further decompose U_A as a product of simpler unitaries and construct a quantum circuit with a limited number of quantum gates. This method of block encoding cannot be easily generalized to matrices of larger dimensions. However, for special matrices with special structures ,e.g. sparse matrices, it is possible to develop some general block encoding strategies which we will discuss in the next section.

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 ϵ . We are now ready to give the definition of block encoding.

24.4.1 The Definition

We introduce a definition of block encoding which we are going to work with. The main idea is rto represent a subnormalized matrix as the upper-left block of a unitary.

$$U = \begin{bmatrix} A/\alpha & * \\ * & * \end{bmatrix} \implies A = \alpha(\langle 0 | \otimes I)U(|0\rangle \otimes I)$$

Definition 24.4.1. (Block encoding) Given an n-qubit matrix A ($N = 2^n$), 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 (α, m, ϵ) -block encoding of A . When the block encoding is exact with $\epsilon = 0$, U_A is called an (α, m) -block encoding of A . The set of all (α, m, ϵ) -block-encoding of A is denoted by $BE_{\alpha,m}(A, \epsilon)$, and we define $BE_{\alpha,m}A = BE(A, 0)$.

Here, m is the number of ancilla bits used to block encode A .

Important Note

In addition to the definition of block-encoding, one can also define an asymmetric version as follows:

$$\|A - \alpha(\langle 0|^{\otimes a} \otimes I)U_A(|0\rangle^{\otimes b} \otimes I)\| \leq \epsilon$$

where a may not equal b . In this case, U_A can be considered to be an $(\alpha, (a, b), \epsilon)$ -or an $(\alpha, \max(a, b), \epsilon)$ block-encoding of A . This can be useful for block-encoding a non-square matrix.

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 establishing $\langle 0^m | \langle i |$ which will be a row vector with 1 in the i th column entry and 0 everywhere else. Thus, upon multiplying the matrices, the $|0^m\rangle |j\rangle$ will pick the j th column of the matrix U_A which will correspond to the j th column of the matrix A and $\langle 0^m | \langle i |$ will pick the i th row of U_A which will correspond to the i th row of A , thus, 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 24.2, and if the ancilla qubits are measured to be 0^m , the system qubits return the normalized state $\sum_i A_{ij} |i\rangle / \|\sum_i A_{ij} |i\rangle\|$.

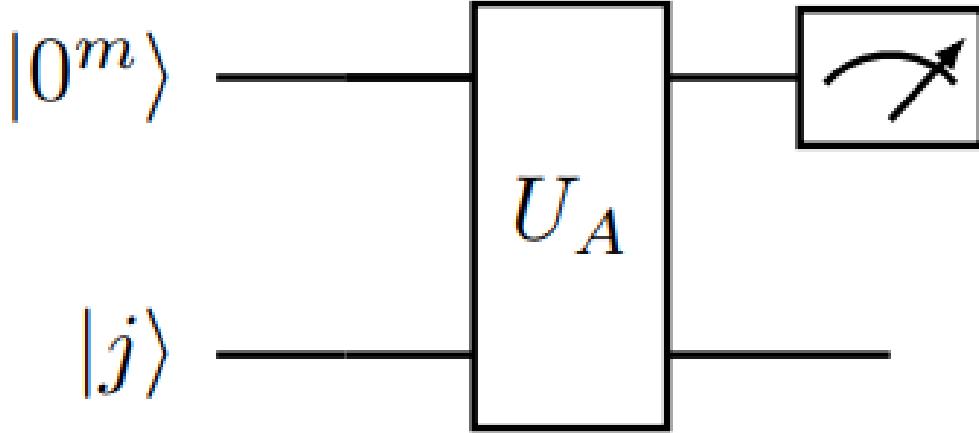


Figure 24.2: Circuit for general block encoding of A

Example 24.4.4. ((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 Σ 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 24.4.5. (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 24.4.6. (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 .

There are several ways of implementing block-encodings based on the choice of matrix. In particular, we will now describe how to construct block-encodings of unitary matrices, density operators, POVM operators, sparse-access matrices, and matrices stored in a QROM (by QROM we mean quantum read-only memory, which stores classical data that can be accessed in superposition). Quantum matrix arithmetics carries out all calculations in an operational way, meaning that the matrices are represented by block-encodings in principle enabling exponential speedups in terms of the dimension of the matrices.

Note that in all these cases since $\|U\| = 1$ we necessarily have $\|A\| \leq \alpha + \epsilon$. Also note that using the above defintiion it seems that we can only represent square

matrices of size $2^n \times 2^n$. However, this is not really a restriction. Suppose that $A \in \mathbb{C}^{n \times m}$, where $n, m \leq 2^s$. Then we can define an embedding matrices denoted by $A_e \in \mathbb{C}^{2^s \times 2^s}$ such that the top-left block of A_e is A and all other elements are zero. This embedding is a faithful representation of the matrices. Suppose that $A, B \in \mathbb{C}^{n \times m}$ are matrices, then $A_e + B_e = (A + B)_e$. Moreover, suppose $X \in \mathbb{C}^{m \times k}$ for some $k \leq 2^s$, then $A_e \cdot C_e = (A \cdot C)_e$.

24.5 Unitary Matrices - Trivial Block Encoding

Unitary matrices are $(1, 0, 0)$ -block-encodings of themselves. Controlled unitaries (e.g. - CNOT) are essentially $(1, 1, 0)$ -block encodings of the controlled operation.

Definition 24.5.1. (Trivial block-encoding) A unitary matrix is a $(1, 0, 0)$ -block encoding of itself.

If we ϵ -approximately implement a unitary U using a ancilla qubits via a unitary \tilde{U} acting jointly on the system and the ancilla qubits, then \tilde{U} is an $(1, a, \epsilon)$ -block encoding of U . This is also a rather trivial encoding. Note that we make a slight distinction between ancilla qubits that are exactly returned to their original state after the computation and the ones that might pick up some error. The latter qubits we will treat as part of the encoding, and the former qubits we usually treat as purely ancillary qubits.

Now we present some non-trivial ways for constructing block-encodings, which will serve as a toolbox for efficiently inputting and representing matrices for arithmetic computations on a quantum computer. We will denote by I_w a w -qubit identity operator, and let $SWAP_w$ denote the swap operation of two w qubit register. We denote by CNOT the controlled not gate that targets the first qubit. When clear from the context we use simply notation $|0\rangle$ to denote $|0\rangle^{\otimes w}$.

24.6 Block encoding of density operators

given an s -qubit density matrix ρ and an $(a + s)$ -qubit unitary G that prepares a purification of ρ as $G|0\rangle^{\otimes a}|0\rangle^{\otimes s} = |\rho\rangle$ (such that $tr_a|\rho\rangle\langle\rho| = \rho$, where tr_a denotes the trace over the first register), then the operator

$$(G^\dagger \otimes I_s)(I_s \otimes SWAP_s)(G \otimes I_s)$$

is a $(1, a+s, 0)$ -block encoding of the density matrix ρ , where I_x denotes the identity operator on a register with x qubits, and $SWAP_s$ denotes the operation that swaps two s -qubit registers.

Lemma 24.6.1. (*Block-encoding of density operators*) Suppose that ρ is an s -qubit density operator and G is an $(a+s)$ -qubit unitary that on the $|0\rangle|0\rangle$ input state prepares a purification $|0\rangle|0\rangle \rightarrow |\rho\rangle$, such that $Tr_a|\rho\rangle\langle\rho| = \rho$. Then $(G^\dagger \otimes I_s)(I_a \otimes SWAP_s)(G \otimes I_s)$ is a $(1, a+s, 0)$ -block encoding of ρ .

Proof. Let r be the Schmidt-rank of ρ , let $\{|\psi_k\rangle : k \in [2^s]\}$ be an orthonormal basis, let $\{|\phi_k\rangle : k \in [r]\}$ be an orthonormal system and let $p \in [0, 1]^{2^s}$ be such that $|\rho\rangle = \sum_{k=1}^r \sqrt{p_k} |\phi_k\rangle |\psi_k\rangle$ and $p_l = 0$ for all $l \in [2^s] \notin [r]$. Then for all $i, j \in [2^s]$ we have that

$$\begin{aligned} & \langle 0 |^{\otimes a+s} \langle \psi_i | (G^\dagger \otimes I_s)(I_a \otimes SWAP_s)(G \otimes I_s) | 0 \rangle^{\otimes a+s} |\psi_j \rangle = \\ &= \langle \rho | \langle \psi_i | (I_a \otimes SWAP_s) |\rho\rangle |\psi_j \rangle \\ &= \left(\sum_{k=1}^r \sqrt{p_k} \langle \phi_k | \langle \psi_k | \right) \langle \psi_i | (I_a \otimes SWAP_s) \left(\sum_{l=1}^r \sqrt{p_l} |\phi_l\rangle |\psi_l\rangle \right) |\psi_j \rangle \\ &= \left(\sum_{k=1}^r \sqrt{p_k} \langle \phi_k | \langle \psi_k | \langle \psi_i | \right) \left(\sum_{l=1}^r \sqrt{p_l} |\phi_l\rangle |\psi_j\rangle |\psi_l\rangle \right) \\ &= \sqrt{p_j p_i} \delta_{ij} \\ &= \langle \psi_i | \rho | \psi_j \rangle \end{aligned}$$

Gilyen recently also showed that an implementation scheme for a POVM operator can also be easily transformed to block-encoding of the POVM operators. By an implementation scheme we mean a quantum circuit u that given input ρ and a ancilla qubits, it sets a flat qubit to 0 with probability $Tr[\rho M]$. \square

24.7 Block encoding of POVM operators

One can construct block-encodings of POVM operators, given access to a unitary that implements the POVM. Specifically, if U is a unitary that implements the POVM M to precision ϵ such that, for all s -qubit density operators ρ we have

$$|Tr(\rho M) - Tr[U(|0\rangle\langle 0|^{\otimes a} \otimes \rho)U^\dagger(|0\rangle\langle 0| \otimes I_{a+s-1})]| \leq \epsilon$$

then $(I_a \otimes U^\dagger(CNOT \otimes I_{a+s-1})(I_a \otimes U))$ is a $(1, 1+a, \epsilon)$ bock encoding of M .

Lemma 24.7.1. (*Block-encoding of POVM operators*) Suppose that U is an $a + s$ qubit unitary, which implements a POVM operator M with ϵ -precision such that for all s -qubit density operator ρ .

$$|Tr[\rho M] - Tr[U(|0\rangle\langle 0|^{\otimes a} \otimes \rho)U^\dagger(|0\rangle\langle 0|^{\otimes 1} \otimes I_{a+s-1})]| \leq \epsilon$$

Then $(I_1 \otimes U^\dagger)(CNOT \otimes I_{a+s-1})(I_a \otimes U)$ is a $(1, 1+a, \epsilon)$ -block encoding of the matrix M .

Proof. First observe that by the cyclicity of trace we have that

$$\begin{aligned} Tr[U(|0\rangle\langle 0|^{\otimes a} \otimes \rho)U^\dagger(|0\rangle\langle 0| \otimes I_{a+s-1})] &= Tr[U(|0\rangle\langle 0|^{\otimes a} \otimes I)\rho(\langle 0|^{\otimes a} \otimes I)U^\dagger(|0\rangle\langle 0| \otimes I_{a+s-1})] \\ &= Tr[\rho(\langle 0|^{\otimes a} \otimes I)U^\dagger(|0\rangle\langle 0| \otimes I_{a+s-1})U(|0\rangle\langle 0|^{\otimes a} \otimes I)] \end{aligned}$$

Together, this implies that for all ρ density operator

$$|Tr[\rho(M - (\langle 0|^{\otimes a} \otimes I)U^\dagger(|0\rangle\langle 0| \otimes I_{a+s-1})U(|0\rangle\langle 0|^{\otimes a} \otimes I))]| \leq \epsilon$$

which is equivalent to saying that $\|M - (\langle 0|^{\otimes a} \otimes I)(U^\dagger(|0\rangle\langle 0| \otimes I_{a+s-1})(I_a \otimes U)(|0\rangle\langle 0|^{\otimes 1+a} \otimes I)\| \leq \epsilon$. We can conclude by observing that

$$\begin{aligned} &(\langle 0|^{\otimes a} \otimes I)U^\dagger(|0\rangle\langle 0| \otimes I_{a+s-1})U(|0\rangle\langle 0|^{\otimes a} \otimes I) = \\ &= (\langle 0|^{\otimes 1+a} \otimes I)(I_a \otimes U^\dagger)(CNOT \otimes I_{a+s-1})(I_1 \otimes U)(|0\rangle\langle 0|^{\otimes 1+a} \otimes I) \end{aligned}$$

□

Now we turn to a more traditional way of constructing block-encoding via state preparation. This is a common technique for example to implement quantum walks. Note that we introduce the notation $[n] - 1$ to denote the set $\{0, 1, \dots, n - 1\}$.

24.8 Block-encoding of Gram matrices

One can implement a block-encoding of a Gram matrix using a pair of state preparation unitaries U_L and U_R . In particular, the product

$$U_A = U_L^\dagger U_R$$

is a $(1, a, 0)$ -block-encoding of the Gram matrix A whose entires are $A_{ij} = \langle \psi_i | \phi_j \rangle$, where

$$U_L |0\rangle^{\otimes a} |i\rangle = |\psi_i\rangle, \quad U_R |0\rangle^{\otimes a} |j\rangle = |\phi_j\rangle$$

Lemma 24.8.1. (*Block encoding of gram matrices by state preparation unitaries*). Let U_L and U_R be “state preparation” unitaries acting on $a + s$ qubits preparing the vectors $\{|\psi_i\rangle : i \in [2^s] - 1\}$, $\{|\phi_j\rangle : j \in [2^s] - 1\}$ such that

$$U_L : |0\rangle|i\rangle \rightarrow |\psi_i\rangle$$

$$U_R : |0\rangle|j\rangle \rightarrow |\phi_j\rangle$$

Then $U = U_L^\dagger U_R$ is an $(1, a, 0)$ -block encoding of the Gram matrix A such that $A_{ij} = \langle \psi_i | \phi_j \rangle$.

One can generalize the above strategy from Gram matrices to arbitrary matrices to produce (α, a, ϵ) -block encodings of general matrices A , where again $\alpha \geq \|A\|$. We now give a few examples of block encodings of more general structured sparse matrices. See block-encoding classical data for details.

24.9 Block Encoding of s-sparse matrix

Sparse-access matrices: given a matrix $A \in \mathbb{C}^{2^w \times 2^w}$ that is s_r -row sparse and s_c -columns sparse (meaning each row/column has at most s_r or s_c nonzero entries), then, defining $\|A\|_{max} = \max_{i,j} |A_{ij}|$, one can create a $(\sqrt{s_r s_c} \|A\|_{max}, w+3, \epsilon)$ -block-encoding of A using oracles O_r , O_c , and O_A , defined below

$$O_r : |i\rangle|k\rangle \rightarrow |i\rangle|r_{ik}\rangle \quad \forall i \in [2^w] - 1, k \in [s_r]$$

$$O_c : |l\rangle|j\rangle \rightarrow |c_{lj}\rangle|j\rangle, \quad \forall j \in [2^w] - 1$$

$$O_A : |i\rangle|j\rangle|0\rangle^{\otimes b} \rightarrow |i\rangle|j\rangle|A_{ij}\rangle \quad \forall i, j \in [2^w] - 1$$

In the above r_{ij} is the index of the j th nonzero entry in the i th row of A (or $j + 2^w$ if there are less than i non zero entries), and c_{ij} is the index of the i th nonzero entry in the j th column of A (or $i + 2^w$ if there are less than j nonzero entries), and $|A_{ij}\rangle$ is a b -bit binary encoding of the matrix element A_{ij} . To build the block-encoding, one needs one query to each of O_r and O_c , and two queries of O_A .

Lemma 24.9.1. (*Block-encoding of sparse-access matrices*). Let $A \in \mathbb{C}^{2^w \times 2^w}$ be a matrix that is s_r row sparse and s_c -column sparse, and each element of A has absolute value at most 1. Suppose that we have access to the following sparse-access oracles acting on two $(w+1)$ qubit registers.

$$O_r : |i\rangle|k\rangle \rightarrow |i\rangle|r_{ik}\rangle \quad \forall i \in [2^w] - 1, k \in [s_r]$$

$$O_c : |l\rangle |j\rangle \rightarrow |c_{lj}\rangle |j\rangle \quad \forall l \in [s_c], j \in [2^w] - 1$$

where r_{ij} is the index for the j th non-zero entry of the i th row of A , or if there are less than i nonzero entries than it is $j + 2^w$, and similarly c_{ij} is the index for the i th non-zero entry of the j column of A , or if there are less than j non zero entries, then it is $i + 2^w$. Additionally assume that we have accesss to an oracle O_A that returns the entries of A in a binary description

$$O_A : |i\rangle |j\rangle |0\rangle^{\otimes b} \rightarrow |i\rangle |j\rangle |a_{ij}\rangle \quad \forall i, j \in [2^w] - 1$$

where a_{ij} is a b -bit binary description (for simplicity we assume here that the binary representation is exact) of the ij matrix element of A . Then we can implement a $(\sqrt{s_r s_c}, w+3, \epsilon)$ block-encoding of A with a single use of O_r, O_c , two uses of O_A and additionally using $O(w + \log^{2.5}(\frac{s_r s_c}{\epsilon}))$ one and two qubit gates while using $\mathcal{O}(b, \log^{2.5}(\frac{s_r s_c}{\epsilon}))$ ancilla qubits.

Proof. We proceed by constructing state preparation unitaries. We will work with 3 registers the first of which is a single qubit register, and the other two registers have $(w+1)$ qubit unitary that implements the map $|0\rangle \rightarrow \sum_{k=1}^s \frac{|k\rangle}{\sqrt{s}}$, it is known that this operator D_s can be implemented with $\mathcal{O}(2)$ quantum gates using $\mathcal{O}(1)$ ancilla qubits. Then we define the $2(w+1)$ qubit unitary $V_L = O_r(I_{w+2} \otimes D_{s_r})SWAP_{w+1}$ such that

$$V_L : |0\rangle^{w+2} |i\rangle \rightarrow \sum_{k=1}^{s_r} \frac{|i\rangle |r_{ik}\rangle}{\sqrt{s_r}} \quad \forall i \in [2^w] - 1$$

We implement the operator $V_R = O_c(D_c \otimes I_{w+1})$ in a similar way acting as

$$V_R : |0\rangle^{w+2} |j\rangle \rightarrow \sum_{l=1}^{s_c} \frac{|c_{lj}\rangle |j\rangle}{\sqrt{s_c}} \quad \forall j \in [2^s] - 1$$

It is easy to see that the above unitaries are such that

$$\langle 0|^{w+2} \langle i| V_L^\dagger V_R |0\rangle^{w+2} |j\rangle = \frac{1}{\sqrt{s_r s_c}} \quad \text{if } a_{ij} \neq 0 \text{ and } 0 \text{ otherwise}$$

Now we define $U_L = I_1 \otimes V_L$ and define U_R as performing the unitary $I_a \otimes V_R$ followed by some extra computation. After performing V_R we et a superposition of index pairs $|i\rangle |j\rangle$. Given an index pair $|i\rangle |j\rangle$ we query the matrix element $|a_{ij}\rangle$ using the oracle O_A . Then we do some elementary computations in order to implement a single qubit gate $|0\rangle \rightarrow a_{ij}|0\rangle + \sqrt{1 - |a_{ij}|^2}|1\rangle$ on the first qubit, with precision $\mathcal{O}(\text{poly}(\frac{\epsilon}{s_r s_c}))$. This can be executed with the stated complexity. Finally we also

need to uncompute everything which requires one more use of O_A . This way we get a good approximation of

$$U_R : |0\rangle^{w+3} |j\rangle \rightarrow \sum_{l=1}^{s_c} \frac{(a_{clj} |0\rangle + \sqrt{1 - |a_{clj}|^2} |1\rangle) |c_{lj}\rangle |j\rangle}{\sqrt{s_c}} \quad \forall j \in [2^w] - 1$$

Note that in the above method the matrix gets subnormalized by a factor of $\frac{1}{\sqrt{s_r s_c}}$. If we would know that for example $\|A\| \leq \frac{1}{2}$, then we could amplify the block-encoding in order to remove this unwanted subnormalization using singular value amplification using block encoding roughly $\sqrt{s_r s_c}$ times. However, under some circumstances one can defeat the subnormalization more efficiently by doing an amplification at the level of state preparation unitaries using a technique called “Uniform spectral gap amplification”. \square

Lemma 24.9.2. (*Preamplified block-encoding of sparse-access matrices*) *Let $A \in \mathbb{C}^{2^w \times 2^w}$ be a matrix that is s_r row sparse and s_c columns sparse, and is given using the input oracles as defined before. let a_i denote the i th row of A and similarly a_j the j th column. Let $q \in [0, 2]$ and suppose that $n_r \in [1, s_r]$ is an upper bound on $\|a_i\| q^q$ and $n_c \in [1, s_c]$ is an upper bound on $\|a_j\|_{2-q}^{2-q}$.*

Let $m = \max[\frac{s_r}{n_r}, s_c n_c]$. Then we can implement a $(\sqrt{\frac{1}{2n_r n_c}}, w+6, \epsilon)$ -block encoding of A with $\mathcal{O}(\sqrt{\frac{s_r}{n_r} \log(\frac{s_r s_c}{\epsilon})})$ uses of O_r , $\mathcal{O}(\sqrt{\frac{s_c}{n_c} \log(\frac{s_r s_c}{\epsilon})})$ uses of O_c , $\mathcal{O}(\sqrt{m} \log(\frac{s_r s_c}{\epsilon}))$ uses of O_A , and additionally using $\mathcal{O}(\sqrt{m} \log(\frac{s_r s_c}{\epsilon}) + \log^{3.5}(\frac{s_r s_c}{\epsilon}))$ one and two qubit gates while using $\mathcal{O}(b, \log^{2.5}(\frac{s_r s_c}{\epsilon}))$ ancilla qubits.

Proof. The idea is very similar to the proof of previous lemma, we implement the unitaries V_L, V_R the same way. However, we define U_R, U_L slightly differently. using a similar method than in previous lemma we implement $\mathcal{O}(\text{poly}(\frac{\epsilon}{s_r s_c}))$ approximations of the maps

$$U_L : |0\rangle^{w+4} |i\rangle \rightarrow \sum_{k=1}^{s_r} \frac{(|a_{ir_{ik}}|^{\frac{q}{2}} |0\rangle + \sqrt{1 - |a_{ir_{ik}}|^q} |1\rangle) |0\rangle |i\rangle |r_{ik}\rangle}{\sqrt{s_r}} \quad \forall i \in [2^w] - 1$$

$$U_R : |0\rangle^{w+4} |j\rangle \rightarrow \sum_{l=1}^{s_c} \frac{\frac{a_{clj}}{|a_{clj}|} |0\rangle (|a_{clj}|^{1-\frac{q}{2}} |0\rangle + \sqrt{1 - |a_{clj}|^{2-q}} |1\rangle) |c_{lj}\rangle |j\rangle}{\sqrt{s_c}} \quad \forall j \in [2^w] - 1$$

It is easy to see that the above unitaries are such that

$$\langle 0|^{w+4} \langle i| U_L^\dagger U_R |0\rangle^{w+4} |j\rangle = \frac{a_{ij}}{\sqrt{s_r s_c}} \quad \forall i, j \in [2^w] - 1$$

We can see that for all $i \in [2^w] - 1$ the modified row vector $\sum_{k=1}^{s_r} \frac{|a_{ir_{ik}}|^{\frac{q}{2}} |0\rangle |0\rangle |i\rangle |r_{ik}\rangle}{\sqrt{s_r}}$ has squared norm at most $\frac{n_r}{s_r}$, and a similar $\frac{n_c}{s_c}$ upper bound holds for the squared norm of the modified column vector. Also observe that

$$(|0\rangle \langle 0| \otimes I_{2w+3})U_L(|0\rangle \langle 0|^{w+4} \otimes I_2) = \sum_{j=0}^{2^w-1} \left(\sum_{k=1}^{s_r} \frac{|a_{ir_{ik}}|^{\frac{q}{2}} |0\rangle |0\rangle |i\rangle |r_{ik}\rangle}{\sqrt{s_r}} \right) \langle 0|^{w+4} \langle j|$$

which is a singular value decomposition with the singular values being the modified row norms. Therefore we can apply singular value amplification with amplification $\gamma_r = \sqrt{\frac{s_r}{\sqrt{2n_r}}}$ and precision $\mathcal{O}(\text{poly}\left(\frac{\epsilon}{s_r s_c}\right))$ approximation of \tilde{U}_L such that

$$(\langle +| \otimes |0\rangle \langle 0| \otimes I_{2w+3})\tilde{U}_L(|+\rangle \otimes |0\rangle \langle 0|^{w+4} \otimes I_w) = \gamma_r \sum_{j=0}^{2^w-1} \left(\sum_{k=1}^{s_r} \frac{|a_{ir_{ik}}|^{\frac{q}{2}} |0\rangle |0\rangle |i\rangle |r_{ik}\rangle}{\sqrt{s_r}} \right) \langle 0|^{w+4} \langle j|$$

Similarly we apply singular value amplification with amplification $\gamma_c = \sqrt{\frac{s_c}{\sqrt{2n_c}}}$ and precision $\mathcal{O}(\text{poly}\left(\frac{\epsilon}{s_r s_c}\right))$ resulting in a $\mathcal{O}(\text{poly}\left(\frac{\epsilon}{s_r s_c}\right))$ approximation of \tilde{U}_R such that

$$\langle ++| \langle 0|^{w+4} \langle i| \tilde{U}_L^\dagger \tilde{U}_R |++\rangle |0\rangle^{2+4} |j\rangle = \gamma_r \gamma_c \frac{a_{ij}}{\sqrt{s_r s_c}} = \frac{a_{ij} \sqrt{2n_r n_c}}{\sqrt{s_r s_c}} \forall i, j \in [2^w] - 1$$

Finally adding 4 Hadamard gates we can change the $|+\rangle$ states above to $|0\rangle$ states, resulting in the $\left(\frac{1}{\sqrt{2n_r n_c}}, w+6, \epsilon\right)$ -block-encoding of A . The complexity statement follows similarly as in the previous proof, with extra observation that the singular value amplifications of U_L and U_R can be performed using degree $\mathcal{O}(\gamma_r \log(\frac{s_r s_c}{\epsilon}))$ and $\mathcal{O}(\gamma_c \log(\frac{s_r s_c}{\epsilon}))$ singular value transformations respectively. \square

For $q \in [0, 2]$ let us define $\mu_q(A) = \sqrt{n_q(A)n_{(2-q)}(A^T)}$, where $n_q(A) = \max_i \|a_i\|_q^q$ is the qth power of the maximum q -norm of the rows of A . Let $A^{(q)}$ denote the matrix of the same dimensions as A , with (for complex values we define these non-integer powers using the principal value of the complex logarithm function) $A_{ij}^{(q)} = \sqrt{a_{ij}^q}$.

Lemma 24.9.3. (*Block-encodings of matrices stored in quantum data structures*) (here we assume that the data structures stores the matrices with sufficient precision. Let $A \in \mathbb{C}^{2^w \times 2^w}$.

1. fix $q \in [0, 2]$. If $A^{(q)}$ and $(A^{(2-q)})^\dagger$ are both stored in quantum accessible data structures. then there exist unitaries U_R and U_L that can be implemented in time $\mathcal{O}(\text{poly}(w \log(1/\epsilon)))$ such that $U_R^\dagger U_L$ is a $(\mu_q(A), w+2, \epsilon)$ block-encoding of A .

2. On the other hand, if A is stored in a quantum accessible data structure, then there exist unitaries U_R and U_L that can be implemented in time $\mathcal{O}(\text{poly}(w \log(e/\epsilon)))$ such that $U_R^\dagger U_L$ is an $(\|A\|_F, w+2, \epsilon)$ -block -encoding of A .

If, in addition to being sparse, the matrix also enjoys some additional structure, e.g. there are only a few distinct values that the matrix elements take, the complexity can be further improved. Finally, note that the sparsity dependence can be essentially quadratically improved to $(\max(s_r, s_c))^{\frac{1}{2}+o(1)}$ using advanced Hamiltonian simulation techniques combined with taking the logarithm of unitaries, however the resulting subroutine may be impractical and comes with a worse precision dependence.

Example 24.9.1. Recall that real symmetric 2×2 matrix example. If we define $\phi_1 = \cos^{-1}(\alpha_1) + \cos^{-1}(\alpha_2)$ and $\phi_2 = \cos^{-1}(\alpha_1) - \cos^{-1}(\alpha_2)$, then we can verify that the block encoding for the 2×2 matrix can be factored as a product of simple unitaries i.e.

$$U_A = U_6 U_5 U_4 U_3 U_2 U_1 U_0$$

where $U_0 = U_6 = I_2 \otimes H \otimes I_2$, $U_1 = R_1 \otimes I_2 \otimes I_2$, $U_2 = U_4 = (I_2 \otimes E_0 + X \otimes E_1) \otimes I_2$, $U_3 = R_2 \otimes I_2 \otimes I_2$, and $U_5 = I_2 \otimes (E_0 \otimes I_2 + E_1 \otimes X)$. Here, H , X are the Hadamard and Pauli-X gates, $R_1 = R_y(\phi_1)$, $R_2 = R_y(\phi_2)$ are the rotation matrices along y axis. and E_0, E_1 are the projectors. The quantum circuit associated with the factorization is given in figure 24.3. Note that in this case, the quantum circuit requires 2 ancilla

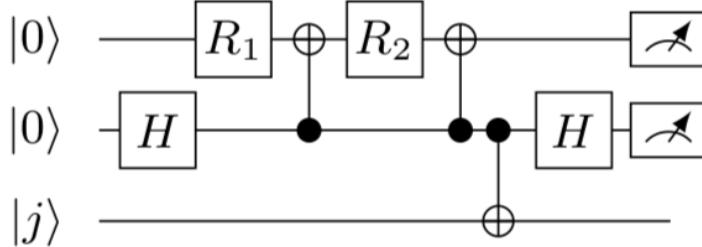


Figure 24.3: A quantum circuit for the block encoding of a 2×2 symmetric matrix A

qubits in addition to the $n = 1$ system qubit required to match the dimension of A , which is $N = 2^n$. As a result the unitary that block encodes the 2×2 matrix A is of dimension 2^3 .

It is possible to develop a general scheme to construct a block encoding and the corresponding quantum circuit for well structured matrices. In particular, when A is sparse with a structured sparsity pattern and the nonzero matrix elements can also be well characterized, an efficient block encoding circuit for A can be constructed.

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 $U_A \in BE_{1,1}(A)$.

Theorem 24.9.4. *Let $c(j, l)$ be a function that gives the row index of the l th (among a list of s) non-zero matrix elements in the j th column of an s -sparse matrix $A \in \mathbb{C}^{N \times N}$ with $N = 2^n$, where $s = 2^m$. If there exists a unitary O_c such that*

$$O_c |l\rangle |j\rangle = |l\rangle |c(j, l)\rangle$$

and a unitary O_A such that

$$O_A |0\rangle |l\rangle |j\rangle = (A_{c(j,l),j} |0\rangle + \sqrt{1 - |A_{c(j,l),j}|^2} |1\rangle) |l\rangle |j\rangle$$

then

$$U_a = (I_2 \otimes D_s \otimes I_N)(I_s \otimes O_c)(I_s \otimes D_s \otimes I_N)$$

block encodes A/s . Here D_s is called a diffusion operator and is defined as

$$D = H \otimes H \otimes \dots \otimes H = H^{\otimes m}$$

Proof. Note that applying D_s to $|0^m\rangle$ yields

$$D_s |0^m\rangle = \frac{1}{\sqrt{s}} \sum_{l=0}^{s-1} |l\rangle$$

where $\{|l\rangle\}$ forms the computational basis in the Hilbert space defined by the m qubits. Our goal is to show that $\langle 0| \langle 0^m | \langle i | U_A | 0 \rangle | 0^m \rangle | j \rangle = A_{ij}/s$. In order to compute the inner product $\langle 0| \langle 0^m | \langle i | U_a | 0 \rangle | 0^m \rangle | j \rangle$, we apply D_s, O_A, O_c to $|0\rangle |0^m\rangle |j\rangle$ successively as illustrated below

$$\begin{aligned} |0\rangle |0^m\rangle |j\rangle &\xrightarrow{I_2 \otimes D_s \otimes I_N} \frac{1}{\sqrt{s}} \sum_{l \in [s]} |0\rangle |l\rangle |j\rangle \\ &\xrightarrow{O_A} \frac{1}{\sqrt{s}} \sum_{l \in [s]} (A_{c(j,l),j} |0\rangle + \sqrt{1 - |A_{c(j,l)}|^2} |1\rangle) |l\rangle |j\rangle \\ &\xrightarrow{I_2 \otimes O_c} \frac{1}{\sqrt{s}} \sum_{l \in [s]} (A_{c(j,l),j} |0\rangle + \sqrt{1 - |A_{c(j,l),j}|^2} |1\rangle) |l\rangle |c(j,l)\rangle \end{aligned}$$

where $[s]$ denotes the set of integers $\{0, 1, \dots, s-1\}$. Instead of multiplying the leftmost factor $I_2 \otimes D_s \otimes I_N$ (note that D_s is Hermitian) to the last line we apply it to $|0\rangle |0^m\rangle |i\rangle$ to obtain

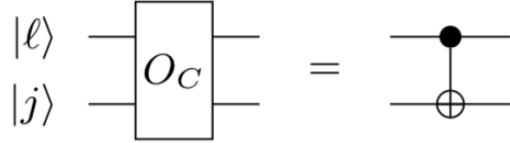
$$|0\rangle |0^m\rangle |i\rangle \xrightarrow{I_2 \otimes D_s \otimes I_N} \frac{1}{\sqrt{s}} \sum_{l' \in [s]} |0\rangle |l'\rangle |i\rangle$$

Finally, taking the inner product between the two results in

$$\langle 0| \langle 0^m | \langle i | U_a | 0 \rangle | 0^m \rangle | j \rangle = \frac{1}{s} \sum_l A_{c(j,l),j} \delta_{i,c(j,l)} = \frac{1}{s} A_{ij}$$

□

The quantum circuit associated with the block encoding is as shown in figure 24.7. Note that the implementation of this block encoding requires $m+1$ ancilla qubits in addition to n system qubits. In order to turn this into an efficient quantum circuit, we need to further decompose the O_c and O_A unitaries into a sequence of quantum gates, which may not be straightforward. In the circuit shown in figure 24.3, O_A is decomposed as $O_A = U_4 U_3 U_2 U_1$ and O_C is simply U_5 . We will now show how these decompositions are constructed systematically.

Figure 24.4: O_c circuit

Although it may be possible to construct a set of controlled quantum gates to achieve O_c and O_A oracles for a specific pair of j and l . This approach will not yield an efficient quantum circuit because the total number of quantum gates in the circuit will be of the order of $\mathcal{O}(N)$, which is exponential with respect to n . Our goal is to construct a circuit that has a gate complexity of $\text{poly}(n)$ i.e. a polynomial in n , at least for certain sparse and/or structured matrices with well defined sparsity patterns.

Note that O_c is unitary, and thus we must have $O_c^\dagger |l\rangle |c(j, l)\rangle = |l\rangle |j\rangle$. This means that for each index $i = c(j, l)$ we can recover the column index j given the value of l . This assumption is of course somewhat restrictive, but it covers important cases such as banded matrices. We remark that it can generally be more difficult to explicitly construct quantum circuits for O_c in these more general query models.

Example 24.9.2. Real Symmetric 2×2 matrix We revisit the matrix considered previously once more. and view it is an s-sparse matrix with $s = 2$ even though it is dense. Therefore we can still use the recipe given in the theorem to construct a block encoding of $A/2$. We will show how explicit circuits for O_c and O_A can be constructed.

1. **The O_c circuit:** Since the second column is a down or upshift of the first one, the function $c(j, l)$ which defines the 0-based row index of the l th nonzero element in the j th column can be defined by

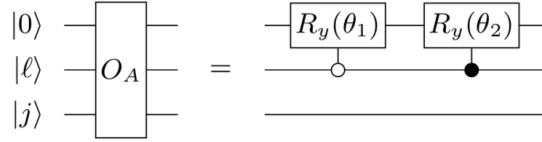
$$c(j, l) = j + l \bmod 2$$

for $j, l = \{0, 1\}$. After enumerating all possible combinations of (l, j) input pairs and their corresponding $(l, c(j, l))$ output pairs as shown in the table 24.1, we see that O_c can simply be implemented as a CNOT gate as shown in the figure 24.4.

l	j	l	$c(j, l)$
0	0	0	0
1	0	1	1
0	1	0	1
1	0	1	0

Table 24.1: All possible output pairs $(l, c(j, l))$ for all possible combinations of (l, j)

2. **The O_A circuit:** The basic strategy to construct a circuit for O_A defined is to use the controlled rotations to place numerical values at proper locations in U_A . Because the matrix A has at most two unique matrix elements. We need two controlled rotations. In general, we need to condition the application of these rotations on the values of j and l . However, since the values of α_1 and α_2 depend solely on l , control is only required on the qubit that takes $|l\rangle$ as the input. We apply the rotation $R_u(\theta_1)$ with $\theta_1 = 2\cos^{-1}(\alpha_1)$ when $l = 0$, and $R_y(\theta_2)$ with $\theta_2 = \cos^{-1}(\alpha_2)$ when $l = 1$. These controlled rotations are combined to yield the following O_A circuit in figure 24.5. Note that the last

Figure 24.5: Implementation of O_A circuit

qubit is not used in this O_A circuit.

ON some quantum hardware it is preferable to use single qubit rotation and CNOT gates in place of controlled rotation gates. These are sometimes referred to as uniformly controlled rotations. In figure 24.6,we show how the controlled rotations used in Q_A circuit can be replaced by an equivalent set of uniformly controlled gates. To see that right hand side is identical to the left hand, we observe first that, if the second qubit is in the $|0\rangle$ state, the circuit on the left applies $R_y(\theta_1)$ to the first qubit. In the circuit on the right, $R_y(\phi_2)R_y(\phi_1)$

Figure 24.6: O_A circuit for uniformly controlled rotations

is applied to the first qubit in this case. These two operations are identical when $\theta_1 = \phi_1 + \phi_2$. Secondly, if the second qubit is in the $|1\rangle$ state, the left circuit applies $R_y(\theta_2)$ to the first qubit, while the right circuit applied the gate sequence $X R_Y(\phi_2) X R_Y(\phi_2)$. It follows from the identity $X R_y(\theta) X = R_Y(-\theta)$ that these operations are equivalent if $\theta_2 = \phi_1 - \phi_2$.

3. The complete circuit: Finally, we substitute circuits, together with $D_s = H$ into the figure 24.7 and obtain the complete block encoding quantum circuit for the matrix. Note that this circuit is identical to the one we have already introduced in figure 24.3.

24.9.1 Banded Circulant Matrix

For a more general s-sparse matrix, WLOG we assume each row and column ha 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 hat 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 ofcourse somewhat restrictive. We shall discuss more general query models.

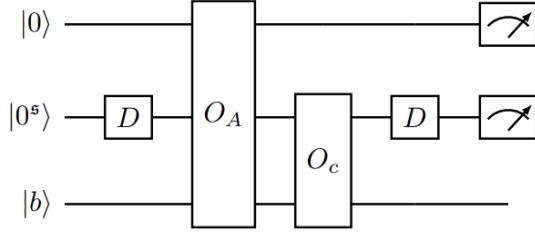
Corresponding to the equation, the matrix entries can be queried via

$$O_A |0\rangle |l\rangle |j\rangle = \left(A_{c(j,l),j} |0\rangle + \sqrt{1 - |A_{c(j,l),j}|^2} |1\rangle \right) |l\rangle |j\rangle$$

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 24.7. The measurement means that to obtain $A |b\rangle$, the ancilla register should return the value 0.

Figure 24.7: Quantum circuit for block encoding an s -sparse matrix

Proposition 24.9.5. *The circuit given in figure 24.7 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

$$\langle 0| \langle 0^p | i | U_A | 0 \rangle | 0^p \rangle | j \rangle = \frac{1}{s} \sum_l A_{c(j,l),j} \delta_{i,c(j,l)} = \frac{1}{s} A_{ij}$$

□

24.10 Hermitian Block Encoding

So far we have considered general s -sparse matrices. Note that if A is Hermitian matrix, its (α, m, ϵ) -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 24.10.1. (Hermitian Block Encoding). Let U_A be an (α, m, ϵ) -block-encoding of A. If U_A is also Hermitian, then it is called an (α, m, ϵ) -Hermitian-block-encoding of A. When $\epsilon = 0$, it is called an (α, m) -Hermitian-block-encoding. The set of all (α, m, ϵ) -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.

24.11 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 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

$$D = I_{n-s} \otimes H^{\otimes s}$$

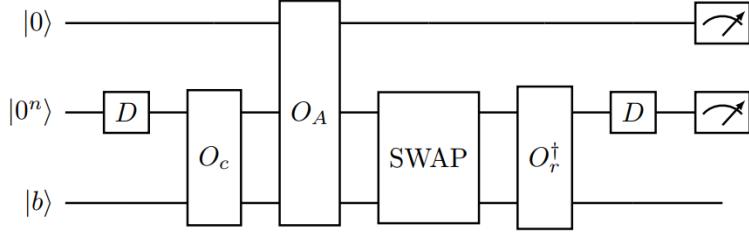


Figure 24.8: Quantum circuit for block encoding of general sparse matrices

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 24.11.1. *The circuit shown in figure 24.8 defines $U_A \in BE_{s,n+1}(A)$.*

Proof. We apply the first four gate sets to the source state

$$\begin{aligned} |0\rangle |0^n\rangle |j\rangle &\xrightarrow{D} \xrightarrow{O_c} \frac{1}{\sqrt{p}} \sum_{l \in [p]} \left(A_{c(j,l),j} |0\rangle + \sqrt{1 - |A_{c(j,l),j}|^2} |1\rangle \right) |c(j,l)\rangle |j\rangle \\ &\xrightarrow{\text{SWAP}} \frac{1}{\sqrt{p}} \sum_{l \in [p]} \left(A_{c(j,l),j} |0\rangle + \sqrt{1 - |A_{c(j,l),j}|^2} |1\rangle \right) |j\rangle |c(j,l)\rangle \end{aligned}$$

we then apply D and O_r to the target state

$$|0\rangle |0^n\rangle |i\rangle \xrightarrow{D} \xrightarrow{O_r} \frac{1}{\sqrt{p}} \sum_{l' \in [p]} |0\rangle |r(i, l')\rangle |i\rangle$$

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 24.8 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

$$\text{SWAP} |a\rangle |i\rangle |b\rangle |j\rangle = |b\rangle |j\rangle |a\rangle |i\rangle$$

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 $\sqrt{A_{ij}} = \sqrt{|A_{ij}|}e^{i\theta_{ij}/2}$.

Proposition 24.11.2. *Figure 24.9 defines $U_A \in HBE_{s,n+2}(A)$.*

Proof. Apply the first four gate sets to the source state gives

$$\begin{aligned} |0\rangle |0^n\rangle |0\rangle |j\rangle &\xrightarrow{D} \xrightarrow{O_c} \frac{1}{\sqrt{s}} \sum_{l \in [s]} |0\rangle |c(j, l)\rangle (\sqrt{A_{c(j,l),j}} |0\rangle \text{sqrt1} - |A_{c(j,l),j} |1\rangle) |j\rangle \\ &\xrightarrow{\text{SWAP}} \frac{1}{\sqrt{s}} \sum_{l \in [s]} \left(\sqrt{A_{c(i,l'),i}} |0\rangle + \sqrt{1 - |A_{c(i,l'),i}|} |1\rangle \right) |i\rangle \end{aligned}$$

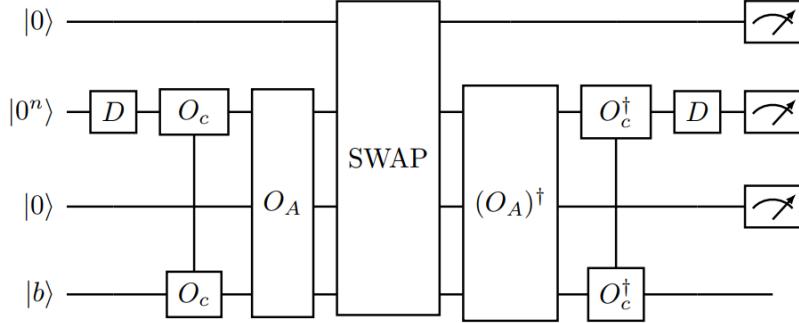


Figure 24.9: Quantum circuit for Hermitian Block encoding of a general Hermitian matrix

Apply the last three gate sets to the target state

$$|0\rangle |0^n\rangle |0\rangle |i\rangle \xrightarrow{D} \xrightarrow{O_c} \xrightarrow{O_A} \frac{1}{\sqrt{s}} \sum_{l \in [s]} |0\rangle |c(i, l')\rangle \left(\sqrt{A_{c(i, l'), i}} |0\rangle + \sqrt{1 - |A_{c(i, l')}|} |1\rangle \right) |i\rangle$$

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 24.9 is essentially construction in. The relation will quantum walks will be discussed further. \square

24.12 Resource cost -gates and qubits

The complexity of block-encoding an operator depends on the type of data or operator being encoded and any underlying assumptions. For instance, unitaries are naturally block-encodings of themselves, and hence their resources requirements depend entirely on their circuit level implementation without any additional overhead for begin a “block-encoding”. By contrast, approaches that make use of state-preparation

and QRAM to implement the block-encoding tend to have larger complexities, as those two subroutines typically dominate the resource requirements, as those two subroutines typically dominate the resource requirements. For example, the best-known circuit that implement block-encoding matrices of classical data for general, dense $N \times N$ matrices uses $\mathcal{O}(N \log(1/\epsilon))$ qubits to achieve minimum T -gate count (which also scales as $\mathcal{O}(N \log(1/\epsilon))$, or a larger $\mathcal{O}(N^2)$ $\mathcal{O}(N^2)$ number of qubits to achieve minimum T -gate depth (which scales as $\mathcal{O}(\log N + \log(1/\epsilon))$). In the sparse-access model, one can use $\mathcal{O}(w + \log^{2.5}(s_r s_c / \epsilon))$ one and two qubit gates, and $\mathcal{O}(b + \log^{2.5}(s_r s_c / \epsilon))$ ancilla qubits, in addition to the calls to the matrix entry O_A and sparse access oracles O_r and O_c , which must be implemented either by computing matrix entries "on the fly" or by using a QRAM primitive. Assuming appropriate binary representations of the numbers A_{ij} , the exponents of the above logarithms can be reduced to 1 using some techniques.

The value of block-encodings is not that it is always cheap to implement them (as it depends on the relevant cost metric and the data access model); rather, the concept of block-encodings is powerful because it allows a practitioner of quantum algorithms to study and optimize the block-encoding construction independently of how it is used within the larger algorithm.

Important Note

A block-encoded matrix A must have norm $\|A\| \leq 1$, or otherwise the matrix must first be normalized, and one must take care to keep track of normalization throughout the computation. In the definition of block-encodings shown above, the parameter α plays the role of normalizing A . Note that often above constructions are suboptimal in the sense that $\alpha \gg \|A\|$, which can lead to increase complexity.

For a given desired block encoding, there can be several independent, yet equally valid implementations, and one can sometimes optimize for various resources when building the block-encoding. For example, many block-encoding strategies require a set in which some classical data is loaded into QRAM, but there are several ways of performing this data-loading step.

When using a block-encoding as part of a larger quantum algorithm, it is important to ensure that the overhead introduced by implementing a block-encoding will not outweigh any potential quantum speedups, as block-encoding can be very resource intensive.

the use of $|0\rangle^{\otimes a}$ as the “signal” state is just one convention - we can use any “signal” state, given a unitary to prepare it. One can also consider a more general definition as “projected unitary encodings” which allows using an arbitrary subspace, rather than just a state-indexed block.

24.13 Example Use Cases

Block-encodings are ubiquitous in quantum algorithms, and they prevail in quantum algorithms that need coherent access to some linear operator or a method of implementing a no-unitary transformation on quantum data. Some specific examples:

- We can manipulate block-encoded operators-for example, take convex or linear combinations, products ,tensor products, and other transformations of an input operator.
- The combination of qubitization with quantum signal processing, or quantum singular value transformation can be used to realize algorithms by applying polynomial transformations to block-encoded matrices. Prominent examples are Hamiltonian simulation via qubitization, and matrix (psuedo) inversion that can be used for solving large linear system of equations or more generally least-squares regression problems.

- Block-encoding can be used to provide coherent access to classical data in a quantum algorithm; for example, loading classical data into a quantum interior point method for portfolio optimization.

Chapter 25

Linear Combination of Unitaries

25.1 Introduction

In practical applications, we are often not interested in constructing the Chebyshev Polynomial of A , but linear combination of Chebyshev polynomials. For instance, the matrix inversion problem can be solved by expanding $f(x) = x^{-1}$ using a linear combination of Chebyshev polynomials on the interval $[-1, -\kappa^{-1}] \cup [\kappa^{-1}, 1]$. Here we assume $\|A\| = 1$ and $\|A\|\|A^{-1}\|$ is the condition number. One way to implement this is via a quantum primitive called the linear combination of unitaries.

Let $T = \sum_{i=0}^{K-1} \alpha_i U_i$ be the linear combination of unitary matrices U_i . For simplicity let $K = 2^a$, and $\alpha_i > 0$ (Without loss of generality we can absorb the phase of α_i into the unitary U_i). Then

$$U = \sum_{i \in [K]} |i\rangle \langle i| \otimes U_i$$

Let V be a unitary operator satisfying

$$V |0^a\rangle = \frac{1}{\sqrt{\|\alpha\|_1}} \sum_{i \in [K]} \sqrt{\alpha_i} |i\rangle$$

and V is called the prepare oracle. The 1-norm of the coefficients is given by

$$\|\alpha\|_1 = \sum_i |\alpha_i|$$

In the matrix form

$$V = \frac{1}{\sqrt{\|\alpha\|_1}} \begin{bmatrix} \sqrt{\alpha_0} & \dots & \dots & \dots \\ \vdots & \dots & \ddots & \dots \\ \sqrt{\alpha_{K-1}} & \dots & \dots & \dots \end{bmatrix}$$

where the first basis is $|0^m\rangle$, and all other basis functions are orthogonal to it. Then

$$V^\dagger = \frac{1}{\sqrt{\|\alpha\|_1}} \begin{bmatrix} \sqrt{\alpha_0} & \dots & \sqrt{\alpha_{K-1}} \\ \dots & \dots & \dots \\ \vdots & \ddots & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

Then T can be implemented using the unitary given as follows (called the LCU lemma).

Lemma 25.1.1. (LCU) Define $W = (V^\dagger \otimes I_n)U(V \otimes I_n)$, then for any $|\psi\rangle$,

$$W |0^a\rangle |\psi\rangle = \frac{1}{\|\alpha\|_1} |0^a\rangle T |\psi\rangle + |\tilde{\perp}\rangle$$

where $|\tilde{\perp}\rangle$ is an unnormalized state satisfying

$$(|0^a\rangle \langle 0^a| \otimes I_n) |\tilde{\perp}\rangle = 0$$

In other words, $W \in BE_{\|\alpha\|_1, a}(T)$

Proof.

$$\begin{aligned}
W |0^a\rangle |\psi\rangle &= (V^\dagger \otimes I_n) U (V \otimes I_n) |0^a\rangle |\psi\rangle \\
(V^\dagger \otimes I_n) U (V \otimes I_n) |0^a\rangle |\psi\rangle &= (V^\dagger \otimes I_n) U \frac{1}{\sqrt{\|\alpha\|_1}} \sum_i \sqrt{\alpha_i} |i\rangle |\psi\rangle \\
&= (V^\dagger \otimes I_n) \left(\sum_j |j\rangle \langle j| \otimes U_j \right) \frac{1}{\sqrt{\|\alpha\|_1}} \sum_i \sqrt{\alpha_i} |i\rangle |\psi\rangle \\
&= (V^\dagger \otimes I_n) \left(\sum_j \sum_i \frac{\sqrt{\alpha_i}}{\sqrt{\|\alpha\|_1}} |j\rangle \langle j|i\rangle \otimes U_j |\psi\rangle \right) \\
&= (V^\dagger \otimes I_n) \left(\frac{1}{\sqrt{\|\alpha\|_1}} \sum_i \sqrt{\alpha_i} |i\rangle \otimes U_i |\psi\rangle \right) \\
&= \frac{1}{\sqrt{\|\alpha\|_1}} (V^\dagger \otimes I_n) \left(\sum_i \sqrt{\alpha_i} |i\rangle \otimes U_i |\psi\rangle \right) \\
&= \frac{1}{\sqrt{\|\alpha\|_1}} \left(\sum_i \sqrt{\alpha_i} V^\dagger |i\rangle \otimes U_i |\psi\rangle \right) \\
&= \frac{1}{\|\alpha\|_1} \left(\sum_i \sqrt{\alpha_i} \begin{bmatrix} \sqrt{\alpha_0} & \dots & \sqrt{\alpha_{K-1}} \\ \vdots & \ddots & \vdots \\ \dots & \dots & \dots \end{bmatrix} |i\rangle \otimes U_i |\psi\rangle \right) \\
&= \frac{1}{\|\alpha\|_1} \left(\sqrt{\alpha_0} \begin{bmatrix} \sqrt{\alpha_0} \\ \vdots \\ \dots \end{bmatrix} \otimes U_0 |\psi\rangle + \dots + \sqrt{\alpha_{K-1}} \begin{bmatrix} \sqrt{\alpha_{K-1}} \\ \vdots \\ \dots \end{bmatrix} \otimes U_{K-1} |\psi\rangle \right) \\
&= \frac{1}{\|\alpha\|_1} \left(\sum_i \left(\begin{bmatrix} \alpha_i \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ \dots \end{bmatrix} \right) \otimes U_i |\psi\rangle \right) \\
&= \frac{1}{\|\alpha\|_1} \left(\sum_i \left(\begin{bmatrix} \alpha_i \\ 0 \\ \vdots \\ 0 \end{bmatrix} + |\perp_i\rangle \right) \otimes U_i |\psi\rangle \right)
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{\|\alpha\|_1} \left(\sum_i (\alpha_i |0^a\rangle + |\perp_i\rangle) \otimes U_i |\psi\rangle \right) \\
&= \frac{1}{\|\alpha\|_1} \left(\sum_i \alpha_i |0^a\rangle \otimes U_i |\psi\rangle + |\perp_i\rangle \otimes U_i |\psi\rangle \right) \\
&= \frac{1}{\|\alpha\|_1} \left(\sum_i \alpha_i |0^a\rangle U_i |\psi\rangle + |\tilde{\perp}_i\rangle \right) \\
&= \frac{1}{\|\alpha\|_1} \left(\sum_i \alpha_i |0^a\rangle U_i |\psi\rangle + \sum_i |\tilde{\perp}_i\rangle \right) \\
&= \frac{1}{\|\alpha\|_1} \left(|0^a\rangle \left(\sum_i \alpha_i U_i \right) |\psi\rangle + |\tilde{\perp}\rangle \right) \\
&= \frac{1}{\|\alpha\|_1} |0^a\rangle T |\psi\rangle + |\tilde{\perp}\rangle
\end{aligned}$$

Thus, upon measuring the first register of a qubits in $|0\rangle$ state will result in the second register being in the state $\frac{1}{\|\alpha\|_1} T |\psi\rangle$ state. \square

The LCU lemma is a useful quantum primitive, as it states that the number of ancilla qubits needed only depends logarithmically on K , the number of terms in the linear combination. Hence it is possible to implement the linear combination of a very large number of terms efficiently. From a practical perspective, the select and prepare oracles uses multi-qubit controls, and can be difficult to implement. If implemented directly, the number of multi-qubit controls again depends linearly on K and is not desirable. Therefore an efficient implementation using LCU (in terms of the gate complexity) also requires additional structures in the prepare and select oracles.

If we apply W to $|0^a|\psi\rangle\rangle$ and measure the ancilla qubits, then the probability of obtaining the outcome 0^a in the ancilla qubits (and therefore obtaining the state $T|\psi\rangle/\|T|\psi\rangle\|$ in the system register) is $(\|T|\psi\rangle\|/\|\alpha\|_1)^2$. The expected number of repetition needed to succeed is $(\|\alpha\|_1/\|T|\psi\rangle\|)^2$. Now we demonstrate that using the amplitude amplification (AA), this number can be reduced to $\mathcal{O}(\|\alpha\|_1/T|\psi\rangle\|)$.

Remark. (Alternate construction of the prepare oracle). In some applications it may not be convenient to absorb the phase of α_i to the select oracle. In such a case, we may modify the prepare oracle instead. If $\alpha_i = r_i e^{i\theta_i}$ with $r_i > 0, \theta_i \in [0, 2\pi)$, we

can define $\sqrt{\alpha_i} = \sqrt{r_i}e^{i\theta_i/2}$, and V is defined as earlier. However instead of V^\dagger , we need to introduce

$$\tilde{V} = \frac{1}{\sqrt{\|\alpha\|_1}} \begin{pmatrix} \sqrt{\alpha_0} & \dots & \sqrt{\alpha_{K-1}} \\ \dots & \dots & \dots \\ \vdots & \ddots & \vdots \\ \dots & \dots & \dots \end{pmatrix}$$

Then following the same proof as earlier, we can show that $W = (\tilde{V} \otimes I_n)U(V \otimes I_n) \in BE_{\|\alpha\|_1,a}(T)$.

Remark. (Linear combination of non-unitaries). Using the block encoding technique, we may immediately obtain linear combination of general matrices that are not unitaries. However, with some abuse of notation, the term "LCU" will be used whether the terms to be combined are unitaries or not. In other words, the term "linear combination of unitaries" should be loosely interpreted as "Linear combination of things" (LCT) in many contexts.

For matrices given as linear combination of unitary operators (LCU), we can block-encode the matrix using the LCU technique. We provide a full description here. For $A = \sum_{i=1}^L c_i V_i$ with V_i unitary, we define the oracles PREPARE (acting on $\lceil \log_2(L) \rceil$ ancilla qubits) and SELECT (acting on the ancilla and register qubits), and implement a $(\sum_i |c_i|, \lceil \log_2(L) \rceil, 0)$ -block-encoding of A , using $U = \text{PREPARE}^\dagger \cdot \text{SELECT} \cdot \text{PREPARE}$. The Hamiltonians of physical system can often be written as a linear combination of a moderate number of Pauli operators leading to a prevalence of this technique in quantum algorithms for chemistry and condensed matter physics.

25.2 Manipulating block-encodings

We use a simple but powerful method for implementing linear combination of unitary operators on a quantum computer. This technique can have exponentially improving the precision of Hamiltonian simulation. Later it was adapted for exponentially improving the precision of quantum linear equation solving. Here we present this method from the perspective of block-encoded matrices.

Given one or more block encodings, we often want to form a single block-encoding of a product, tensor product, or linear combination of the individual block-encoded operators. This can be achieved as outlined below, using additional ancilla qubits.

We will consider the case of two operators A and B with straightforward generalizations to additional operators. We are given an (α, a, ϵ_a) block encoding U_A of

A , and a (β, b, ϵ_b) block-encoding U_B of B . operators A and B act on system qubits s .

First we define state preparation unitaries in order to conveniently state our result in the following lemma.

Definition 25.2.1. (State preparation pair) Let $y \in \mathbb{C}^m$ and $\|y\|_a \leq \beta$, the pair of unitaries (P_L, P_R) is called a (β, b, ϵ) -state preparation pair if $P_L |0\rangle^{\otimes b} = \sum_{j=0}^{2^b-1} c_j |j\rangle$ and $P_R |0\rangle^{\otimes b} = \sum_{j=1}^{2^b-1} d_j |j\rangle$ such that $\sum_{j=0}^{m-1} |\beta(c_j^* d_j) - y_j| \leq \epsilon$ and for all $j \in m, \dots, 2^b - 1$ we have $c_j^* d_j = 0$.

Now we show how to implement a block-encoding of block-encoded operators.

25.2.1 Products

In general if we want to take the product of two block encoded matrices we need to treat their ancilla qubits separately. In this case as the following lemma shows the errors simply add up and the block encoding does not introduce any additional errors.

Lemma 25.2.1. (*Product of block-encoded matrices*) If U is an (α, a, δ) -block-encoding of an s qubit operator A , and V is an (β, b, ϵ) -block-encoding of an s qubit operator B then $(I_b \otimes U)(I_a \otimes V)$ is an $(\alpha\beta, a + b, \alpha\epsilon + \beta\delta)$ -block-encoding of AB .

Proof.

$$\begin{aligned} & \|AB - \alpha\beta((\langle 0|^{\otimes a+b} \otimes I)(I_b \otimes U)(I_a \otimes V)(|0\rangle^{\otimes a+b}) \text{ otimes } I)\| \\ &= \|AB - \alpha(\langle 0|^{\otimes a} \otimes I)U(|0\rangle^{\otimes a} \otimes I)\beta(\langle 0|^{\otimes b} \otimes I)V(|0\rangle^{\otimes b} \otimes I)\| \\ &= \|AB - \tilde{A}B + \tilde{A}B - \tilde{A}\tilde{B}\| \\ &= \|(A - \tilde{A})B + \tilde{A}(B - \tilde{B})\| \\ &\leq \|A\tilde{A}\|\beta + \alpha\|B - \tilde{B}\| \\ &\leq \alpha\epsilon + \beta\delta \end{aligned}$$

where $\tilde{A} = \alpha(\langle 0|^{\otimes a} \otimes I)U(|0\rangle^{\otimes a} \otimes I)$ and $\tilde{B} = \beta(\langle 0|^{\otimes b} \otimes I)V(|0\rangle^{\otimes b} \otimes I)$. \square

In the special case when the encoded matrices are unitaries and their block-encoding does not use any extra scaling factor, then we might reuse the ancilla qubits, however it introduces an extra error term, which can be bounded by geometric mean of the two input error bounds.

Lemma 25.2.2. (*Product of block-encoded matrices*) If U is an $(1, a, \delta)$ -block-encoding of an s -qubit operator A , and V is an $(1, a, \epsilon)$ -block-encoding of an s -qubit operator B then UV is a $(1, a, \delta + \epsilon + 2\sqrt{\delta\epsilon})$ -block-encoding of the unitary operator AB .

Proof. It is enough to show that for all s -qubit pure states $|\phi\rangle, |\psi\rangle$ we have that

$$|\langle\phi|AB|\psi\rangle - \langle\phi|(\langle 0|^{\otimes a} \otimes I)UV(|0\rangle^{\otimes a} \otimes I)|\psi\rangle| \leq \delta + \epsilon + 2\sqrt{\delta\epsilon}$$

Observe that

$$\begin{aligned} & \langle\phi|(\langle 0|^{\otimes a} \otimes I)UV(|0\rangle^{\otimes a})|\psi\rangle \\ &= \langle\phi|(\langle 0|^{\otimes a} \otimes I)U((|0\rangle\langle 0|^{\otimes a} \otimes I) + ((I - |0\rangle\langle 0|^{\otimes a}) \otimes I)) + ((I - |0\rangle\langle 0|^{\otimes a} \otimes I)V(|0\rangle^{\otimes a} \otimes I)|\psi\rangle \\ &= \langle\phi|(\langle 0|^{\otimes a})U(|0\rangle^{\otimes a} \otimes I)(\langle 0|^{\otimes a} \otimes I)V(|0\rangle^{\otimes a} \otimes I)|\psi\rangle \\ &+ \langle\phi|(\langle 0|^{\otimes a} \otimes I)U((I - |0\rangle\langle 0|^{\otimes a}) \otimes I)V(|0\rangle^{\otimes a} \otimes I)|\psi\rangle \end{aligned}$$

Now we can see that similarly, we have

$$\begin{aligned} & |\langle\phi|AB|\psi\rangle - \langle\phi|(\langle 0|^{\otimes a} \otimes I)(\langle 0|^{\otimes a} \otimes I)V(|0\rangle^{\otimes a} \otimes I)|\psi\rangle| \\ &= |\langle\phi|(AB - (\langle 0|^{\otimes a} \otimes I)U(|0\rangle^{\otimes a} \otimes I)(\langle 0|^{\otimes a} \otimes I)V(|0\rangle^{\otimes a} \otimes I))|\psi\rangle| \\ &\leq \|AB - (\langle 0|^{\otimes a} \otimes I)U(|0\rangle^{\otimes a} \otimes I)(\langle 0|^{\otimes a} \otimes I)V(|0\rangle^{\otimes a} \otimes I)\| \\ &\leq \delta + \epsilon \end{aligned}$$

Finally note that

$$\begin{aligned} & |\langle\phi|(\langle 0|^{\otimes a} \otimes I)U(I - |0\rangle\langle 0|^{\otimes a}) \otimes I)V(|0\rangle^{\otimes a} \otimes I)|\psi\rangle| \\ &= |\langle\phi|(\langle 0|^{\otimes a} \otimes I)U((I - |0\rangle\langle 0|^{\otimes a}) \otimes I)^2V(|0\rangle^{\otimes a} \otimes I)|\psi\rangle| \\ &\leq \|((I - |0\rangle\langle 0|^{\otimes a}) \otimes I)U(|0\rangle^{\otimes a} \otimes I)|\phi\rangle\| \cdot \|((I - |0\rangle\langle 0|^{\otimes a}) \otimes I)V(|0\rangle^{\otimes a} \otimes I)|\psi\rangle\| \\ &= \sqrt{1 - \|(|0\rangle\langle 0|^{\otimes a} \otimes I)U(|0\rangle^{\otimes a} \otimes I)|\phi\rangle\|^2} \cdot \sqrt{1 - \||0\rangle\langle 0|^{\otimes a} \otimes I)V(|0\rangle^{\otimes a} \otimes I)V(|0\rangle^{\otimes a} \otimes I)|\psi\rangle\|^2} \\ &\leq \sqrt{1 - (1 - \delta)^2} \cdot \sqrt{1 - (1 - \epsilon)^2} \\ &\leq 2\sqrt{\delta\epsilon} \end{aligned}$$

□

The following corollary suggest that if we multiply together multiple block-encoded unitaries, the error may grow super-linearly, but it increases at most quadratically with the number of factors in the product.

Corollary 25.2.3. (*Product of multiple block-encoded unitaries*) Suppose that U_j is an $(1, a, \epsilon)$ block encoding of an s qubit unitary operator W_j for all $j \in [K]$. Then $\prod_{j=1}^K U_j$ is an $(1, a, 4K^2\epsilon)$ -block-encoding of $\prod_{j=1}^K W_j$.

Proof. First observe that for the product of two matrices we get the precision bound 4ϵ by the above lemma. If $K = 2^k$ for some $k \in \mathbb{N}$. Then we can apply the above observation in a recursive fashion in a binary tree structure, to get the upper bound $4^k\epsilon$ on the precision, and observe that $4^k = K^2$.

If $2^{k-1} \leq K \leq 2^k$ we can just add identity operators so that we have 2^k matrices to multiply, which gives the precision bound $4^k\epsilon \leq 4^{1+\log_2 K}\epsilon = 4K^2\epsilon$. \square

The operator $U_{AB} = (I_b \otimes U_A)(U_B \otimes I_a)$ is an $(\alpha\beta, a+b, \alpha\epsilon_b + \beta\epsilon_a)$ -block-encoding of AB , where I_x denotes the identity operator on x qubits. For example, if $a = b$, this construction uses twice as many ancilla qubits for block-encoding the product compared to the block-encoding of individual matrices. In fact we can assume without loss of generality that $a = b$ (by taking the max of the two) and improve the construction using the circuit in figure ??.

25.2.2 Tensor products

the operation $U_{A \otimes B} = U_a \otimes U_B$ is an $(\alpha\beta, a + b, \alpha\epsilon_b + \beta\epsilon_a)$ -block-encoding of the operator $A \otimes B$.

25.2.3 Linear Combinations

Linear combinations of block-encodings can be viewed as a generalization of the linear combination of Unitaries (LCU). We wish to implement a block-encoding of $\sum_{i=0}^{L-1} c_i A_i$, where $c_i \in \mathbb{R}$ (the LCU trick can also be extended to complex coefficients) and define $\lambda = \sum_{i=0}^{L-1} |c_i|$. We consider L block-encodings U_i that are $(1, m, \epsilon_i)$ -block encodings of A_i . We note that in cases where the block-encodings have different α_i or m_i values, the former can be absorbed into the c_i values and latter can be taken as $m = \max_i m_i$.

We first define an operator PREPARE by the following action on $|0^{\lceil \log_2(L) \rceil}\rangle$

$$\text{PREPARE } |0^{\lceil \log_2(L) \rceil}\rangle = \frac{1}{\sqrt{\lambda}} \sum_j \sqrt{|c_j|} |j\rangle$$

that prepares a weighted superposition on an ancilla register, such that the amplitudes are proportional to the square roots of the absolute values of the desired

coefficients. We also define

$$SELECT = \sum_{j=0}^{L-1} |j\rangle\langle j| \otimes sign(c_j)U_j$$

We then have the following result:

$$(\langle 0^{\lceil \log_2(L) \rceil} | \otimes I) PREPARE^\dagger \cdot SELECT \cdot PREPARE(|0^{\lceil \log_2(L) \rceil}\rangle \otimes I) = \frac{1}{\lambda} \sum_{l=0}^{L-1} c_l U_l$$

i.e. $U_{LC} = PREPARE^\dagger \cdot SELECT \cdot PREPARE$ is a $(\lambda, \lceil \log_2(L) \rceil, 0)$ -block-encoding of the LCU $\sum_I c_i U_i$. This is the standard LCU trick, and it does not require U_i to be block encodings (or we can view them as $(1, 0, 0)$ block-encodings of themselves). This technique can be used in Hamiltonian simulation, or to instantiate a block-encoding.

If, as specified above, U_i are block-encodings of \tilde{A}_i (which approximate A_i), we also have the following result:

$$\left\| \left(\sum_i c_i A_i \right) - \lambda (\langle 0^{m+\lceil \log_2(L) \rceil} | \otimes I) U_{LC} (|0^{m+\lceil \log_2(L) \rceil}\rangle \otimes I) \right\| \leq \sum_i |c_i| \epsilon_i$$

Hence, U_{LC} is a $(\lambda, \lceil \log_2(L) \rceil + m, \lambda \max_i \epsilon_i)$ block-encoding of $\sum_i c_i A_i$.

Lemma 25.2.4. (*Linear combination of block-encoded matrices*) *Let $A = \sum_{j=1}^m y_j A_j$ be an s qubit operator and $\epsilon \in \mathbb{R}_+$. Suppose that (P_L, P_R) is a (β, b, ϵ_1) state preparation pair for y . $W = \sum_{j=0}^{m-1} |j\rangle\langle j| \otimes U_j + ((I - \sum_{j=0}^{m-1} |j\rangle\langle j|) \otimes I_a \otimes I_s)$ is an $s+a+b$ qubit unitary such that for all $j \in 0, \dots, m$ we have that U_j is an (α, a, ϵ_2) block-encoding of A_j . Then we can implement a $(\alpha\beta, a+b, \alpha\epsilon_1 + \alpha\beta\epsilon_2)$ -block-encoding of A , with a single use of W , P_r and P_L^\dagger .*

Proof. Observe that $\tilde{W} = (P_L^\dagger \otimes I_a \otimes I_s) W (P_R \otimes I_a \otimes I_s)$ is an $(\alpha\beta, a+b, \alpha\epsilon_1 + \alpha\beta\epsilon_2)$ -block-encoding of A .

$\alpha\beta\epsilon_2$)block-encoding of A .

$$\begin{aligned}
 \|A - \alpha\beta(\langle 0|^{\otimes b} \otimes \langle 0|^{\otimes a} \otimes I)\tilde{W}(|0\rangle^{\otimes b} \otimes |0\rangle^{\otimes a} \otimes I)\| &= \|A - \alpha \sum_{j=0}^{m-1} \beta(c_j^* d_j)(\langle 0|^{\otimes a} \otimes I)U_j(|0\rangle^{\otimes a} \otimes I)\| \\
 &\leq \alpha\epsilon_1 + \|A - \alpha \sum_{j=0}^{m-1} y_j(\langle 0|^{\otimes a} \otimes I)U_j(|0\rangle^{\otimes a} \otimes I)\| \\
 &\leq \alpha\epsilon_1 + \alpha \sum_{j=0}^{m-1} y_j \|A_j - (\langle 0|^{\otimes a} \otimes I)U_j(|0\rangle^{\otimes a} \otimes I)\| \\
 &\leq \alpha\epsilon_1 + \alpha \sum_{j=0}^{m-1} y_j \epsilon_2 \\
 &\leq \alpha\epsilon_1 + \alpha\beta\epsilon_2
 \end{aligned}$$

□

25.3 Examples

Linear combinations of block-encodings are used to obtain mixed-parity functions in QSVT required for Hamiltonian simulation. LCU trick is used for Hamiltonian simulation, or to instantiate block-encodings of chemistry or condensed matter physics Hamiltonians.

Example 25.3.1. (Linear combination of two matrices). Let U_A, U_B be two n-qubit unitaries, and we would like to construct a block encoding of $T = U_A + U_B$.

There are two terms in total, so one ancilla qubit is needed. The prepare oracle needs to implement

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

so this is the Hadamard gate. The circuit is given by figure 25.1, which constructs $W \in BE_{\sqrt{2},1}(T)$. A special case is the linear combination of two block encoded matrices. Given two n-qubit matrices A, B , for simplicity let $U_A \in BE_{1,m}(A), U_B \in BE_{1,m}(B)$. We would like to construct a block encoding of $T = A + B$. The circuit is given by figure 25.2, which constructs $W \in BE_{\sqrt{2},1+m}(T)$. This is also an example of a linear combination of non-unitary matrices.

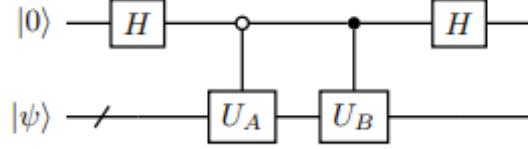


Figure 25.1: Circuit for linear combination of two unitaries

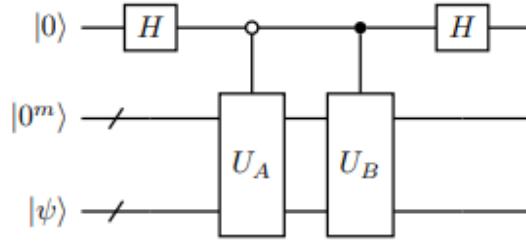


Figure 25.2: Circuit for linear combination of two block encoded matrices

Example 25.3.2. (Transverse field Ising Model) Consider the following TFIM model with periodic boundary conditions ($Z_n = Z_0$) and $n = 2^m$.

$$\hat{H} = - \sum_{i \in [n]} Z_i Z_{i+1} - \sum_{i \in [n]} X_i$$

In order to use LCU we need $(m+1)$ ancilla qubits. The prepare oracle can be simply constructed from the Hadamard gate

$$V = H^{\otimes(m+1)}$$

and the select oracle implements

$$U = \sum_{i \in [n]} |i\rangle \langle i| \otimes (-Z_i Z_{i+1} + \sum_{i \in [n]} |i+n\rangle \langle i+n| \otimes (-X_i))$$

The corresponding $W \in BE_{\sqrt{2n}, n+1}(\hat{H})$.

Example 25.3.3. (Block encoding of a matrix polynomial) Let us use the LCU lemma to construct the block encoding for an arbitrary matrix polynomial for a Hermitian matrix A .

$$f(A) = \sum_{k \in [K]} \alpha_k T_k(A)$$

with $\|\alpha\|_1 = \sum_{k \in [K]} |\alpha_k|$ and we set $K = 2^a$. For simplicity assume $\alpha_k \geq 0$.

We have constructed $U_k = (U_A Z_\Pi)^k$ as the $(1, m)$ block encoding of $T_k(A)$. From each U_k we can implement the select oracle

$$U = \sum_{k \in [K]} |k\rangle \langle k| \otimes U_k$$

via multi-qubit controls. Also given the availability of the prepare oracle

$$V |0^a\rangle = \frac{1}{\sqrt{\|\alpha\|_1}} \sum_{k \in [K]} \sqrt{\alpha_k} |k\rangle$$

we obtain a $(\|\alpha\|_1, m + a)$ -block-encoding of $f(A)$.

The need of using a ancilla qubits, and even more importantly the need to implement the prepare and select oracles is undesirable. We will later see that the quantum signal processing (QSP) and quantum singular value transformation (QSVT) can drastically reduce both sources of difficulties.

Example 25.3.4. (Matrix functions given by a matrix Fourier Series). Instead of block encoding, LCU can also utilize a different query model based on Hamiltonian simulation. Let A be an n -qubit Hermitian matrix. Consider $f(x) \in \mathbb{R}$ given by its Fourier expansion (up to a normalization factor)

$$f(x) = \int \hat{f}(x) e^{ikx} dk$$

and we are interested in computing the matrix function via numerical quadrature

$$f(A) = \int \hat{f}(k) e^{ikA} dk \approx \Delta k \sum_{k \in [K]} \hat{f}(k) e^{ikA}$$

Here K is a uniform grid discretizing the interval $[-L, L]$ using $|K| = 2^p$ grid points, and the grid space is $\Delta k = 2L/|K|$. The prepare oracle is given by the coefficients $c_k = \Delta k \hat{f}(k)$, and the corresponding subnormalization factor is

$$\|c\|_1 = \sum_{k \in [K]} \Delta k |\hat{f}(k)| \approx \int |\hat{f}(k)| dk$$

The select oracle is

$$U = \sum_{k \in K} |k\rangle \langle k| \otimes e^{ikA}$$

This can be efficiently implemented using the controlled matrix powers as in figure 21.3, where the basic unit is the short time Hamiltonian simulation $e^{i\Delta k A}$. This can be used to block encode a large class of matrix functions.

Important Note

Performing linear algebraic manipulations of block-encodings using these primitives can quickly increase the ancilla count of the algorithm and worsen the normalization factor of the block-encoding. Amplifying a subnormalized block-encoding is possible, but costly, requiring an amount of time scaling roughly linearly in the amplification factor. Given a single block-encoded operator A , the above primitives can be used to implement a block-encoding polynomial in A . However, this can be achieved using quantum singular value transformation.

Chapter 26

Matrix functions of Hermitian Matrices

Let A be an n-qubit Hermitian matrix. Then A has the eigenvalue decomposition

$$A = V\Lambda V^\dagger$$

Here $\Lambda = \text{diag}(\{\lambda_i\})$ is a diagonal matrix, and $\lambda_0 \leq \dots, \lambda_{N-1}$. Let the scalar function f be well defined on all λ_i 's. then the matrix function $f(A)$ can be defined on the eigendecomposition:

Definition 26.0.1. (Matrix functions of Hermitian matrices) Let $A \in \mathbb{C}^{N \times N}$ be a Hermitian matrix with eigenvalue decomposition $V\Lambda V^\dagger$. Let $f : \mathbb{R} \rightarrow \mathbb{C}^{N \times N}$ be a scalar function such that $f(\lambda_i)$ is defined for all $i \in [N]$. The matrix function is defined as

$$f(A) = Vf(\Lambda)V^\dagger$$

where

$$f(\Lambda) = \text{diag}(f(\lambda_0), f(\lambda_1), \dots, f(\lambda_{N-1}))$$

This chapter introduces an efficient quantum circuit to compute $f(A)|b\rangle$ for any state $|b\rangle$. Throughout the discussion we assume A is queried in the block encoding model denoted by U_A . For simplicity we assume that there is no error in the block encoding, i.e., $U_A \in BE_{\alpha,m}(A)$, and without the loss of generality we can take $\alpha = 1$.

Many tasks in scientific computation can be expressed in terms of matrix functions. Here are a few examples:

- Hamiltonian simulation: $f(A) = e^{\iota At}$

- Gibbs state preparation: $f(A) = e^{-\beta A}$
- Solving linear systems of equation $f(A) = A^{-1}$
- Eigenstate filtering $f(A) = \mathbb{I}_{(-\infty, 0)}(A - \mu I)$

A key technique for representing matrix functions is called qubitization.

Chapter 27

Qubitization

27.0.1 Motivation

Qubitization has the following motivation: we are given block-encoding of U_A of a Hermitian operator A , and we wish to manipulate A - e.g., implement A^2 , or more generally some function $f(A)$. However, the eigenvalues of U_A are typically unrelated to those of A , and plain repeated applications of U_A do not in general produce the desired behavior. Qubitization converts the block-encoding U_A into a unitary operator W (sometimes called a qubiterate or a qubitized quantum walk operator) having the following guaranteed advantageous properties:

1. W is a block-encoding of the operator A .
2. The spectrum of W has a nice relation to the spectrum of A .
3. Repeated applications of W leads to structured behavior that can be cleanly analyzed.

This combination of features means that qubitization can be used for applying polynomial transformations to the spectrum of A . For example, repeated application of W implements Chebyshev polynomials of A , while other polynomials can be implemented by using quantum signal processing.

The key observation is that a qubitization unitary W has eigenvalues and eigenvectors that relate in a nice way to those of A . Thus one can also perform quantum phase estimation on W to learn these quantities, providing a potentially cheaper alternative to such tasks compared to approaches based on explicitly Hamiltonian simulation for implementing $U = e^{\iota At}$

27.0.2 Overview

We are given a $(1, m, 0)$ -block-encoding U_A of Hermitian A such that

$$A = (\langle 0^m | \otimes I) U_A (|0^m\rangle \otimes I) \implies U_A = \begin{pmatrix} A & \cdot \\ \cdot & \cdot \end{pmatrix}$$

where $|0^m\rangle$ denotes $|0\rangle^{\otimes m}$. First we will assume U_A is also Hermitian (implying $U_A^2 = I$, where I is the identity matrix). Let A have spectral decomposition $A = \sum_{\lambda} \lambda |\lambda\rangle \langle \lambda|$. An application of U_A to an eigenstate $|\lambda\rangle$ of A gives

$$U_A |0^m\rangle \otimes |\lambda\rangle = \lambda |0^m\rangle |\lambda\rangle + \sqrt{1 - \lambda^2} |\perp_{0^m, \lambda}\rangle$$

where $|\perp_{0^m, \lambda}\rangle$ is a state perpendicular to $|0^m\rangle$ (If $\lambda = \pm 1$, then there is no need for $|\perp_{0^m, \lambda}\rangle$, and the subspace S_{λ} becomes one dimensional). Noting $U_A^2 = I$ reveals that the $2D$ subspace S_{λ} spanned by $\{|0^m\rangle |\lambda\rangle, |\perp_{0^m, \lambda}\rangle\}$ is invariant under the action of U_A . U_A is restricted onto S_{λ} can be described by the matrix

$$\begin{pmatrix} \lambda & \sqrt{1 - \lambda^2} \\ \sqrt{1 - \lambda^2} & -\lambda \end{pmatrix}$$

which is a $2D$ reflection with the eigenvalues ± 1 . Clearly, repeated application of (self-inverse) U_A can have limited effect on any input state. Qubitization uses a reflection $Z_{|0^m\rangle} = (2|0^m\rangle \langle 0^m| - I)$ to transform U_A into a Grover-like operator $W = Z_{|0^m\rangle} U_A$ which has the following matrix when restricted onto the invariant subspace S_{λ} in the $\{|0^m\rangle |\lambda\rangle, |\perp_{0^m, \lambda}\rangle\}$ basis

$$[W]_{(|0^m\rangle |\lambda\rangle, |\perp_{0^m, \lambda}\rangle)} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \lambda & \sqrt{1 - \lambda^2} \\ \sqrt{1 - \lambda^2} & -\lambda \end{pmatrix} = \begin{pmatrix} \lambda & \sqrt{1 - \lambda^2} \\ -\sqrt{1 - \lambda^2} & \lambda \end{pmatrix}$$

showing that W is still a $(1, m, 0)$ -block-encoding of A . This has the form of a Y -axis rotation

$$[W]_{|0^m\rangle |\lambda\rangle, |\perp_{0^m, \lambda}\rangle} = \begin{pmatrix} \cos(\theta_{\lambda}) & \sin(\theta_{\lambda}) \\ -\sin(\theta_{\lambda}) & \cos(\theta_{\lambda}) \end{pmatrix}$$

where $\theta_{\lambda} = \arccos(\lambda)$. Therefore, W has eigenvalues $e^{\pm \arccos(\lambda)}$ with respective eigenvectors $(|0^m\rangle |\lambda\rangle \pm i|0^m, \lambda\rangle)/\sqrt{2}$, which can be accessed using quantum phase estimation.

Furthermore, we can see that on the space of the subspace S_λ repeated application of W acts as

$$\begin{aligned} W^d &= \bigoplus_{\lambda} \begin{pmatrix} \cos(d\theta_\lambda) & \sin(d\theta_\lambda) \\ -\sin(d\theta_\lambda) & \cos(d\theta_\lambda) \end{pmatrix} \\ &= \bigoplus_{\lambda} \begin{pmatrix} T_d(\lambda) & \sqrt{1-\lambda^2}U_{d-1}(\lambda) \\ -\sqrt{1-\lambda^2}U_{d-1}(\lambda) & T_d(\lambda) \end{pmatrix} \\ &= \begin{pmatrix} T_d(A) & \cdot \\ \cdot & \cdot \end{pmatrix} \end{aligned}$$

where $T_d(\cdot)$ and $U_d(\cdot)$ are degree- d Chebyshev polynomials of the first and second kind, respectively. Therefore, W^d applies the polynomial transformation T_d to each eigenvalue of A thereby implementing $T_d(A)$.

27.1 Qubitization of Hermitian matrices with Hermitian block encoding

We first introduce some heuristic idea behind qubitization. For any $-1 < \lambda \leq 1$, we can consider a 2×2 rotation matrix,

$$O(\lambda) = \begin{pmatrix} \lambda & -\sqrt{1-\lambda^2} \\ \sqrt{1-\lambda^2} & \lambda \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

where we have performed the change of variable $\lambda = \cos \theta$ with $0 \leq \theta < \pi$

Now direct computations show that

$$O^k(\lambda) = \begin{pmatrix} \cos(k\theta) & -\sin(k\theta) \\ \sin(k\theta) & \cos(k\theta) \end{pmatrix}$$

Using the definition of Chebyshev Polynomials (of first and second kinds, respectively)

$$T_k(\lambda) = \cos(k \cos^{-1} \lambda), \quad U_{k-1}(\lambda) = \frac{\sin k\theta}{\sin \theta} = \frac{\sin(k \cos^{-1} \lambda)}{\sqrt{1-\lambda^2}}$$

we have

$$O^k(\lambda) = \begin{pmatrix} T_k(\lambda) & -\sqrt{1-\lambda^2}U_{k-1}(\lambda) \\ \sqrt{1-\lambda^2}U_{k-1}(\lambda) & T_k(\lambda) \end{pmatrix}$$

Note that if we somehow replace λ by A , we have immediately obtain a $(1, 1)$ block encoding for the Chebyhsev polynomial $T_k(A)$! This is precisely what qubitization aims at achieving, though there are some small twists.

In the simplest scenario we assume that $U_a \in HBE_{1,m}(A)$. Start from the spectral decomposition

$$A = \sum_i \lambda_i |v_i\rangle \langle v_i|$$

we have that for each eigenstate $|v_i\rangle$,

$$U_A |0^m\rangle |v_i\rangle = |0^m\rangle A |v_i\rangle + |\tilde{\perp}_i\rangle = \lambda_i |0^m\rangle |v_i\rangle + |\tilde{\perp}_i\rangle$$

Here $|\tilde{\perp}_i\rangle$ is an unnormalized state that is orthogonal to all states of the form $|0^m\rangle |\psi\rangle$, i.e.,

$$\Pi |\tilde{\perp}\rangle = 0$$

where

$$\Pi = |0^m\rangle \langle 0^m| \otimes I_n$$

is a projection operator.

Since the right hand side of equation is an unnomralized state, we may also write

$$|\tilde{\perp}_i\rangle = \sqrt{1 - \lambda_i^2} |\perp_i\rangle$$

where $|\perp_i\rangle$ is a normalized state.

Now if $\lambda_i = \pm 1$, then $\mathcal{H}_i = \text{span}\{|0^m\rangle |v_i\rangle\}$ is already an invariant subspace of U_A , and $|\perp_i\rangle$ can be any state. Otherwise, use the fact that $U_A = U_A^\dagger$, we can apply U_A again to both sides of the equation and obtain

$$\begin{aligned} U_A^2 |0^m\rangle |v_i\rangle &= U_A(\lambda_i |0^m\rangle |v_i\rangle + |\tilde{\perp}_i\rangle) \\ |0^m\rangle |v_i\rangle &= \lambda_i U_A |0^m\rangle |v_i\rangle + U_A \sqrt{1 - \lambda_i^2} |\perp_i\rangle \\ &= \lambda_i (\lambda_i |0^m\rangle |v_i\rangle + |\tilde{\perp}_i\rangle) + \sqrt{1 - \lambda_i^2} U_A |\perp_i\rangle \\ &= \lambda_i^2 |0^m\rangle |v_i\rangle + \lambda_i \sqrt{1 - \lambda_i^2} |\perp_i\rangle + \sqrt{1 - \lambda_i^2} U_A |\perp_i\rangle \end{aligned}$$

$$|0^m\rangle |v_i\rangle - \lambda_i^2 |0^m\rangle |v_i\rangle = \sqrt{1 - \lambda_i^2} (\lambda_i |\perp_i\rangle + U_A |\perp_i\rangle)$$

$$\frac{1 - \lambda_i^2}{\sqrt{1 - \lambda_i^2}} |0^m\rangle |v_i\rangle = \lambda_i |\perp_i\rangle + U_A |\perp_i\rangle$$

$$U_A |\perp_i\rangle = \sqrt{1 - \lambda_i^2} |0^m\rangle |v_i\rangle - \lambda_i |\perp_i\rangle$$

Therefore $\mathcal{H}_i = \text{span}\{|0^m\rangle|v_i\rangle, |\perp_i\rangle\}$ is an invariant subspace of U_A . Furthermore, the matrix representation of U_A with respect to the basis $\mathcal{B}_i = \{|0^m\rangle|v_i\rangle, |\perp_i\rangle\}$ is

$$[U_A]_{\mathcal{B}_i} = \begin{pmatrix} \lambda_i & \sqrt{1 - \lambda_i^2} \\ \sqrt{1 - \lambda_i^2} & -\lambda_i \end{pmatrix}$$

i.e., U_A restricted to \mathcal{H}_i is a reflection operator. This also leads to the name “qubitization”, which means that each eigenvector $|v_i\rangle$ is “qubitized” into a two-dimensional space \mathcal{H}_i .

In order to construct a block encoding for $T_k(A)$, we need to turn U_A into a rotation. For this note that \mathcal{H}_i is also an invariant subspace for the projection operator Π :

$$\Pi |0^m\rangle|v_i\rangle = (|0^m\rangle\langle 0^m| \otimes I_n)(|0^m\rangle \otimes |v_i\rangle) = |0^m\rangle \otimes |v_i\rangle = |0^m\rangle|v_i\rangle$$

and

$$\Pi |\tilde{\perp}_i\rangle = 0$$

Thus,

$$[\Pi]_{\mathcal{B}_i} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Similarly define $Z_\Pi = 2\Pi - 1$, since

$$[Z_\Pi]_{\mathcal{B}_i} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Z_Π acts as a reflection operator restricted to each subspace \mathcal{H}_i . Then \mathcal{H}_i is the invariant subspace for the iterate

$$O = U_A Z_\Pi$$

and

$$[O]_{\mathcal{B}_i} = \begin{pmatrix} \lambda_i & -\sqrt{1 - \lambda_i^2} \\ \sqrt{1 - \lambda_i^2} & \lambda_i \end{pmatrix}$$

is the desired rotation matrix. Therefore

$$[O^k]_{\mathcal{B}_i} = [(U_A Z_\Pi)^k]_{\mathcal{B}_i} = \begin{pmatrix} T_k(\lambda_i) & -\sqrt{1 - \lambda_i^2} U_{k-1}(\lambda_i) \\ \sqrt{1 - \lambda_i^2} U_{k-1}(\lambda_i) & T_k(\lambda_i) \end{pmatrix}$$

Since $\{|0^m\rangle|v_i\rangle\}$ spans the range of Π , we have

$$O^k = \begin{pmatrix} T_k(A) & * \\ * & * \end{pmatrix}$$

i.e., $O^k = (U_A Z_\Pi)^k$ is a $(1, m)$ -block encoding of the Chebyshev polynomial $T_k(A)$.

In order to implement Z_Π , note that if $m = 1$, then Z_Π is just the Pauli Z gate. When $m > 1$, the circuit

returns $|1\rangle|0^m\rangle$ if $b = 0^m$, and $-|1\rangle|b\rangle$ if $b \neq 0^m$. So this precisely implements Z_Π where the signal qubit $|1\rangle$ is used as a work register. We may also discard the signal qubit, and resulting unitary is denoted by Z_Π .

In other words, the circuit in figure implements the operator O . Repeating the circuit k times gives the $(1, m + 1)$ -block encoding of $T_k(A)$.

Remark. (Alternative perspective of qubitization) The fact that an arbitrarily large block encoding matrix U_A can be partially block diagonalized into N subblocks of size 2×2 may seem a rather peculiar algebraic structure. In fact there are other alternative perspectives and derivations of the qubitization result. Some noticeable ones include the use of Jordan's Lemma, and the use of cosine-sine (CS) decomposition.

27.2 Qubitization of hermitian matrices with general block encoding

Previously we assumed that $U_A = U_A^\dagger$ to block encode a Hermitian matrix A . For instance, a sparse Hermitian matrices, such Hermitian block encoding can be constructed following the recipe as done in the previous section. However, this can come at the expense of requiring additional structures and oracles. In general, the block encoding of a Hermitian matrix may not be hermitian itself. In this section we demonstrate that the strategy of qubitization can be modified to accommodate general block encodings.

Again start from the eigen decomposition $A = \sum_i \lambda_i |\lambda_i\rangle\langle\lambda_i|$, we apply U_A to $|0^m\rangle|v_i\rangle$ and obtain

$$U_A |0^m\rangle|v_i\rangle = \lambda_i |0^m\rangle|v_i\rangle + \sqrt{1 - \lambda_i^2} |\perp'_i\rangle$$

where $|\perp_i\rangle$ is a normalized state satisfying $\Pi|\perp'_i\rangle = 0$

Since U_A block encodes a Hermitian matrix A , we have

$$U_A^\dagger = \begin{pmatrix} A & * \\ * & * \end{pmatrix}$$

which implies that there exist another normalized state $|\perp_i\rangle$ satisfying $\Pi|\perp_i\rangle = 0$ and

$$U_A^\dagger |0^m\rangle|v_i\rangle = \lambda_i |0^m\rangle|v_i\rangle + \sqrt{1 - \lambda_i^2} |\perp_i\rangle$$

Now apply U_A to both sides, we obtain

$$|0^m\rangle |v_i\rangle = \lambda_i^2 |0^m\rangle |v_i\rangle + \lambda_i \sqrt{1 - \lambda_i^2} |\perp'_i\rangle + \sqrt{1 - \lambda_i^2} U_A |\perp_i\rangle$$

which gives

$$U_A |\perp_i\rangle = \sqrt{1 - \lambda_i^2} |0^m\rangle |v_i\rangle - \lambda_i |\perp'_i\rangle$$

Define

$$\mathcal{B}_i = \{|0^m\rangle |v_i\rangle, |\perp_i\rangle\}, \quad \mathcal{B}'_i = \{|0^m\rangle |v_i\rangle, |\perp'_i\rangle\}$$

and the associated two-dimensional subspaces $\mathcal{H}_i = \text{span}(\mathcal{B}_i)$, $\mathcal{H}'_i = \text{span}(\mathcal{B}'_i)$, we find that U_A maps \mathcal{H}_i to \mathcal{H}'_i . Correspondingly U_A^\dagger maps \mathcal{H}'_i to \mathcal{H}_i .

Then in the matrix representation we get,

$$[U_A]_{\mathcal{B}'_i}^{\mathcal{B}'_i} = \begin{pmatrix} \lambda_i & \sqrt{1 - \lambda_i^2} \\ \sqrt{1 - \lambda_i^2} & -\lambda_i \end{pmatrix}$$

Similar calculations show that

$$[U_A^\dagger]_{\mathcal{B}'_i}^{\mathcal{B}_i} = \begin{pmatrix} \lambda_i & \sqrt{1 - \lambda_i^2} \\ \sqrt{1 - \lambda_i^2} & -\lambda_i \end{pmatrix}$$

Meanwhile both \mathcal{H}_i and \mathcal{H}'_i are the invariant subspace of the projector Π , with matrix representation

$$[\Pi]_{\mathcal{B}_i} = [\Pi]_{\mathcal{B}'_i} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Therefore

$$[Z_\Pi]_{\mathcal{B}_i} = [Z_\Pi]_{\mathcal{B}'_i} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Hence \mathcal{H}_i is an invariant subspace of $\tilde{O} = U_A^\dagger Z_\Pi U_A Z_\Pi$, with matrix representation

$$[\tilde{O}]_{\mathcal{B}_i} = \begin{pmatrix} \lambda_i & -\sqrt{1 - \lambda_i^2} \\ \sqrt{1 - \lambda_i^2} & \lambda_i \end{pmatrix}^2$$

repeating k times, we have

$$[\tilde{O}^k]_{\mathcal{B}_i} = (U_A^\dagger Z_\Pi U_A Z_\Pi)^k = \begin{pmatrix} \lambda_i & -\sqrt{1 - \lambda_i^2} \\ \sqrt{1 - \lambda_i^2} & \lambda_i \end{pmatrix}^{2k}$$

Since any vector $|0^m\rangle |\psi\rangle$ can be expanded in terms of the eigenvectors $|0^m\rangle |v_i\rangle$, we have

$$(U_A^\dagger Z_\Pi U_A Z_\Pi)^k = \begin{pmatrix} T_{2k}(A) & * \\ * & * \end{pmatrix}$$

Therefore if we would like to construct an even order Chebyshev polynomial $T_{2k}(A)$, the circuit $(U_A Z_\Pi U_A Z_\Pi)^k$ straightforwardly gives a $(1, m)$ -block-encoding.

In order to construct the block-encoding of an odd polynomial $T_{2k+1}(A)$, we note that

$$[U_A Z_\Pi (U_A^\dagger Z_\Pi U_A Z_\Pi)^k]_{\mathcal{B}'_i} = \begin{pmatrix} T_{2k+1}(\lambda_i) & -\sqrt{1-\lambda_i^2} U_{2k}(\lambda_i) \\ \sqrt{1-\lambda_i^2} U_{2k}(\lambda_i) & T_{2k+1}(\lambda_i) \end{pmatrix}$$

Using the fact that $\mathcal{B}_i, \mathcal{B}'_i$ share the common basis $|0^m\rangle |v_i\rangle$, we still have the block-encoding

$$U_A Z_\Pi (U_A^\dagger Z_\Pi U_A Z_\Pi)^k = \begin{pmatrix} T_{2k+1}(A) & * \\ * & * \end{pmatrix}$$

Therefore $U_A Z_\Pi (U_A^\dagger Z_\Pi U_A Z_\Pi)^k$ is a $(1, m)$ -block encoding of $T_{2k+1}(A)$.

In summary, the block encoding of $T_l(A)$ is given by applying $U_A Z_\Pi$ and $U_A^\dagger Z_\Pi$ alternatively. If $l = 2k$, then there are exactly k such pairs. The quantum circuit for each pair \tilde{O} is

Otherwise if $l = 2k + 1$, then there is an extra $U_A Z_\Pi$. The effect is to map each eigenvector $|v\rangle |v_i\rangle$ back and forth between the two-dimensional subspaces \mathcal{H}_i and \mathcal{H}'_i . We shall later see that the separate treatment even/odd polynomials will play a more prominent role.

Now that we have obtained $O_k \in BE_{1,m}(T_k(A))$ for all k , we can use the LCU again to construct block encodings for linear combination of Chebyshev polynomials.

27.3 Dominant resource cost - qubits and gates

The resource cost of qubitization is inherited from the cost of the block-encoding. Given a Hermitian $(\alpha, m, 0)$ -block-encoding, the qubitization operator \circledast is a (non-Hermitian) $(\alpha, m, 0)$ -block-encoding, and it uses no additional qubits. The operation $Z_{|0^m\rangle} = 2(|0^m\rangle \langle 0^m| - I)$ can be implemented (up to a global phase) with an m -qubit controlled Z gate, equivalent to an m -qubit Toffoli gate up to single-qubit gates. An example qubitization circuit is shown below in figure ?? for $m = 3$. Implementing a block-encoding of a degree d Chebyshev polynomial applied to A requires d calls to U_A and $Z_{|0^m\rangle}$.

if the block-encoding of U_A is not Hermitian, qubitization can be achieved using the construction that uses one additional qubit and one call to controlled U_A and controlled U_A^\dagger to implement the Hermitian block-encoding.

$$U'_A = ((HX) \otimes I)(|0\rangle\langle 0| \otimes U_A + |1\rangle\langle 1| \otimes U_A^{\text{dagger}})(H \otimes I)$$

An alternate to qubitization is based on singular value transformation that uses the sequence $Z_{|0^m\rangle}U_A^\dagger Z_{|0^m\rangle}U_A$, analogous to the earlier W^2 , acting as

$$\begin{pmatrix} \lambda & \sqrt{1-\lambda^2} \\ -\sqrt{1-\lambda^2} & \lambda \end{pmatrix}$$

on a 2D subspace analogous to S_λ . The approach can be extended to odd-degree polynomials with a single additional application of $Z_{|0^m\rangle}U_A$. The advantage of this approach is it does not require U_A to be Hermitian, thus there is no need for an additional qubit or calls to controlled $U_A^{\pm 1}$. This approach may be referred to as “quantum eigenvalue transformation” as this is a special case of quantum singular value transformation just applied to Hermitian A .

Important Note

The original formulation of qubitization discussed above requires a Hermitian or normal block-encoded matrix A . The concept can be extended to general (non-square) matrices via quantum singular value transformation, providing a significant generalization, however in some cases quantum signal processing and its generalized versions can exploit additional structure that comes for example from the extra symmetries of Hermitian block-encoding leading to potential cost factor savings.

Consider for example Hamiltonian simulation, where QSVT separately implements $\sin(tH)$ and $\cos(tH)$ using a block-encoding of U_H of the Hamiltonian H , and applies a 3 step oblivious amplification procedure on top of linear combination of unitaries to implement $\exp(itH)$. Meanwhile, quantum signal processing implements $\exp(itH)$ directly but requires an additional ancilla qubit and controlled access to a Hermitian block encoding U_H^\dagger , which, when implemented, uses both controlled U_H and U_H^\dagger resulting in a factor of ≈ 4 overhead. Altogether these considerations suggest that the QSVT-based approach might have a slightly better constant factor overhead, particularly when controlled U_H is significantly more costly to implement than U_H . If U_H is already hermitian then quantum signal processing can have an improved complexity.

27.4 Example - Use cases

- Some quantum algorithms in quantum chemistry that compute energies perform phase estimation on a qubitization operator W implemented via calls to a block-encoding of the electronic structure Hamiltonian. This avoids the approximation error incurred when performing phase estimation on $e^{\iota Ht}$, implemented via Hamiltonian simulation.
- Qubitization acts as a precursor to quantum singular value transformation, which extends the concept to general matrices and unifies it with quantum signal processing.

Chapter 28

Quantum Eigenvalue Transformation

We know how to construct a block encoding of an Hermitian matrix A (the block encoding matrix itself can be non-Hermitian), use qubitization to block encode $T_k(A)$, use LCU to block encode an arbitrary polynomial function of A (up to a subnormalization factor). This framework is conceptually appealing, but the practical implementation of the select and prepare oracles are by no means straightforward, and can come at significant costs.

28.1 Hermitian Block encoding

Note that $\iota Z = e^{\iota \frac{\pi}{2} Z}$, if A is given by a Hermitian block encoding U_A , the block encoding of the Chebyshev polynomial can be written

$$O^d = (-\iota)^d \prod_{j=1}^d (U_A e^{\iota \frac{\pi}{2} Z_\Pi})$$

This is a special case of the following representation. Note that $(-\iota)^d$ is an irrelevant phase factor and can be discarded.

$$U_{\tilde{\Phi}} = e^{\iota \tilde{\phi}_0 Z_\Pi} \prod_{j=1}^d (U_A e^{\iota \tilde{\phi}_j Z_\Pi})$$

The representation is called a Quantum Eigenvalue Transformation (QET).

Due to qubitization, $U_{\tilde{\phi}}$ should block encode some matrix polynomial of A . We first state the following theorem without proof.

Theorem 28.1.1. (*Quantum signal processing, a simplified version*) Consider

$$U_A = \begin{pmatrix} x & \sqrt{1-x^2} \\ \sqrt{1-x^2} & -x \end{pmatrix}$$

For any $\tilde{\Phi} = (\tilde{\phi}_0, \dots, \tilde{\phi}_d) \in \mathbb{R}^{d+1}$,

$$U_{\tilde{\Phi}} = e^{\iota \tilde{\phi}_0 Z} \prod_{j=1}^d (U_A e^{\iota \tilde{\phi}_j Z}) = \begin{pmatrix} P(x) & * \\ * & * \end{pmatrix}$$

where $P \in \mathbb{C}[x]$ satisfy

1. $\deg(P) \leq d$
2. P has parity ($d \bmod 2$)
3. $|P(x)| \leq 1 \forall x \in [-1, 1]$.

Also define $-\tilde{\Phi} = (-\tilde{\phi}_0, \dots, -\tilde{\phi}_d) \in \mathbb{R}^{d+1}$, then

$$U_{-\tilde{\Phi}} = e^{-\iota \tilde{\phi}_0 Z} \prod_{j=1}^d (U_A e^{-\iota \tilde{\phi}_j Z}) = U_{\tilde{\Phi}}^* = \begin{pmatrix} P^*(x) & * \\ * & * \end{pmatrix}$$

Here $P^*(x)$ is the complex conjugation of $P(x)$, and $U_{\tilde{\Phi}}^*$ is the complex conjugation (without transpose) of $U_{\tilde{\Phi}}$.

Remark. This theorem can be proved inductively. However, this is a special case of the quantum signal processing, so we will omit the proof here. IN fact, we will also state the converse of the result, which describes precisely the class of matrix polynomials that can be described by such phase factor modulations. The condition (1) states that the polynomial degree is upper bounded by the number of U_A 's and the condition (3) is simply a consequence of that $U_{\tilde{\Phi}}$ is a unitary matrix. The condition (2) is less obvious, but should not come at a surprise, since we have seen the need ot treating even and odd polynomials separately in the case of qubitization with a general block encoding can be proved directly by taking the complex conjugation of $U_{\tilde{\Phi}}$.

Following the qubitization procedure we immediately have the following theorem.

Theorem 28.1.2. (Quantum Eigenvalue transformation with Hermitian block encoding) Let $U_A \in HBE_{1,m}(A)$. Then for any $\tilde{\Phi} = (\tilde{\phi}_0, \dots, \tilde{\phi}_d) \in \mathbb{R}^{d+1}$,

$$U_{\tilde{\Phi}} = e^{\iota \tilde{\Phi}_0 Z_\Pi} \prod_{j=1}^d (U_A e^{\iota \tilde{\Phi}_j Z_\Pi}) = \begin{pmatrix} P(A) & * \\ * & * \end{pmatrix} \in BE_{1,m}(P(A))$$

where $P \in \mathbb{C}[x]$ satisfies the requirements in the previous theorem.

Using this Theorem, we may construct the block encoding of a matrix polynomial without invoking LCU. The cost is essentially the same as block encoding a Chebyshev polynomial.

In order to implement $e^{\imath\phi Z_{\Pi}}$, we note that the quantum circuit denoted by CR_{ϕ} is in figure 28.1 returns $e^{\imath\phi} |0\rangle|0^m\rangle$ if $b = 0^m$, and $e^{-\imath\phi} |0\rangle|b\rangle$ if $b \neq 0^m$. So omitting the signal qubit, this is precisely $e^{\imath\phi Z_{\Pi}}$. Therefore, if A is given by a Hermitian

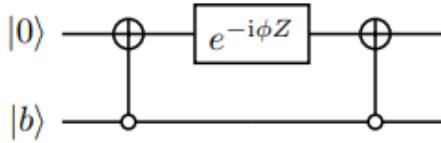


Figure 28.1: Implementing the controlled rotation circuit for quantum eigenvalue transformation

block encoding U_A , we can follow the argument in previous sections and construct the following unitary. The corresponding quantum circuit is in figure 28.2, which uses one extra ancilla qubit. When measuring the $(m + 1)$ -ancilla qubits and obtain $|0\rangle|0^m\rangle$ the corresponding (unnormalized) state in the system register is $P(A)|\psi\rangle$. The QET described by the circuit in figure 28.2 generally construct a block encoding of $P(A)$ for some complex polynomial P . In practical applications (such as those later in this chapter) we would like to construct a block encoding of $P_{Re}(A) = (ReP)(A) = \frac{1}{2}(P(A) + P^*(A))$ instead. Below we demonstrate that a simple modification of figure 28.2 allows us to achieve this goal. To this end, we use equation

$$U_{-\{\tilde{P}hi}} = e^{-\iota \tilde{\Phi}_0 Z_\Pi} \prod_{j=1}^d (U_A e^{-\iota \tilde{\Phi}_j Z_\Pi}) = \begin{pmatrix} P^*(A) & * \\ * & * \end{pmatrix}$$

Qubitization allows us to construct. So all we need is to negate all phase factors in $\tilde{\Phi}$. In order to implement $CR_{-\phi}$, we do not actually need to implement a new circuit.

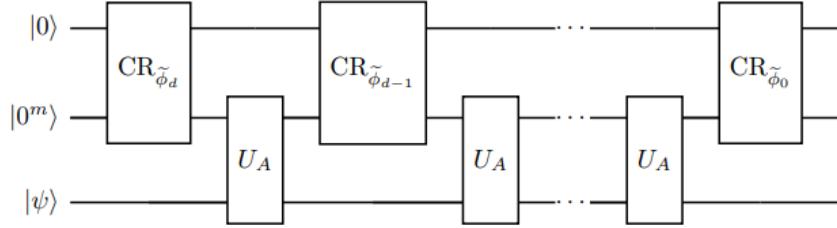
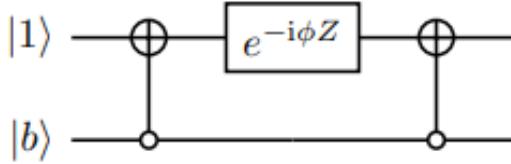


Figure 28.2: Circuit of quantum eigenvalue transformation to construct $U_{P(A)} \in BE_{1,m+1}(P(A))$, using $U_A \in HBE_{1,m}(A)$



Instead we may simply change the signal qubit from $|0\rangle$ to $|1\rangle$. Now we claim the circuit in figure 28.3 implements a block encoding $U_{P_{Re}(A)} \in BE_{1,m=1}(P_{Re}(A))$. This circuit can be viewed as an implementation of the linear combination of unitaries $\frac{1}{2}(U_{P^*(A)} + U_{P(A)})$. To verify this, we may evaluate

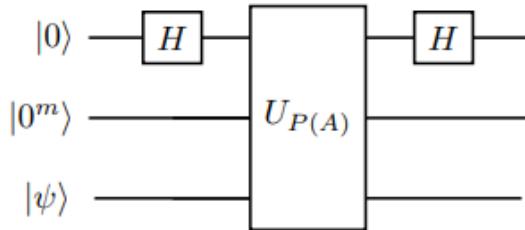


Figure 28.3: Circuit of quantum eigenvalue transformation for constructing a $(1, m + 1)$ -block encoding of $P_{Re}(A)$

$$\begin{aligned}
|0\rangle |0^m\rangle |\psi\rangle &\xrightarrow{H \otimes I_{m+n}} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) |0^m\rangle |\psi\rangle \\
&\xrightarrow{U_{P(A)}} \frac{1}{\sqrt{2}}|0\rangle (|0^m\rangle P(A) |\psi\rangle + |\perp\rangle) + \frac{1}{\sqrt{2}}|1\rangle (|0^m\rangle P^*(A) |\psi\rangle + |\perp'\rangle) \\
&\xrightarrow{H \otimes I - m + n} |0\rangle \left(|0^m\rangle \frac{P(A) + P^*(A)}{2} |\psi\rangle \right) + |\tilde{\perp}\rangle \\
&= |0\rangle |0^m\rangle P_{Re}(A) |\psi\rangle + |\tilde{\perp}\rangle
\end{aligned}$$

here $|\perp\rangle$ and $|\perp'\rangle$ are two $(m+n)$ -qubit state orthogonal to any state $|0^m\rangle |x\rangle$, while $|\tilde{\perp}\rangle$ is a $(m+n+1)$ - qubit state orthogonal to any state $|0\rangle |0^m\rangle |x\rangle$. In other words, by measuring all $(m+1)$ ancilla qubits and obtain 0^{m+1} , the corresponding (unnormalized) state in the system register is $P_{Re}(A) |\psi\rangle$.

28.1.1 General Block Encoding

If A is given by a general block encoding U_A , the quantum eigenvalue transformation should consist of an alternating sequence of U_A , U_A^\dagger gates. The circuit is given by figure 28.4, and the corresponding block encoding is described in the following theorem. Note that the Hermitian block encoding becomes a special case with $U_A = U_A^\dagger$.

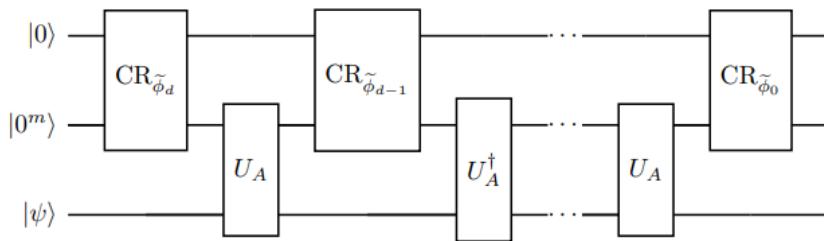


Figure 28.4: Circuit of quantum eigenvalue transformation to construct $U_{P(A)} \in BE_{1,m+1}(P(A))$, using $U_A \in BE_{1,m}(A)$. Here U_A, U_A^\dagger should be applied alternatively. When d is even, the last U_A gate should be replaced by U_A^\dagger .

Theorem 28.1.3. (Quantum eigenvalue transformation with general block encoding) Let $U_A \in BE_{1,m}(A)$. Then for any $\tilde{\Phi} = (\tilde{\phi}_0, \dots, \tilde{\phi}_d) \in \mathbb{R}^{d+1}$, let

$$U_{\tilde{\Phi}} = e^{\iota \tilde{\phi}_0 Z_\Pi} \prod_{j=1}^{d/2} [U_A^\dagger e^{\iota \tilde{\phi}_{2j-1} Z_\Pi} U_A e^{\iota \tilde{\phi}_{2j+1} Z_\Pi}]$$

when d is even, and

$$U_{\tilde{\Phi}} = (-\iota)^d e^{\iota \tilde{\phi}_0 Z_{\Pi}} (U_A e^{\iota \tilde{\Phi}_1 Z_{\Pi}}) \prod_{j=1}^{(d-1)/2} [U_A^\dagger e^{\iota \tilde{\phi}_{2j} Z_{\Pi}} U_A e^{\iota \tilde{\phi}_{2j+1} Z_{\Pi}}]$$

when d is odd. Then

$$U_{\tilde{\Phi}} = \begin{pmatrix} P(A) & * \\ * & * \end{pmatrix} \text{ in } BE_{1,m}(P(A)),$$

where $P \in \mathbb{C}[x]$ satisfy the condition in the previous theorem.

Following exactly the same procedure, we find that the circuit in figure with $U_{P(A)}$ is given by figure 28.3 implements a $U_{P_{Re}(A)} \in BE_{1,m+1}(P_{Re}(A))$.

28.1.2 General Matrix Polynomials

In practical applications, we may be interested in matrix polynomials $f(A)$, where $f(x) \in \mathbb{R}[x]$ does not have a definite parity. This violates the parity requirement of Theorem. This can be solved using the LCU technique. Note that

$$f(x) = f_{even}(x) + f_{odd}(x)$$

where $f_{even}(x) = \frac{1}{2}(f(x) + f(-x))$, $f_{odd}(x) = \frac{1}{2}(f(x) - f(-x))$. If $|f(x)| \leq 1$ on $[-1, 1]$, then $|f_{even}(x)|, |f_{odd}(x)| \leq 1$ on $[-1, 1]$, and $f_{even}(x), f_{odd}(x)$ can be each constructed using the circuit in figure. Introducing another ancilla qubit and using the LCU technique we obtain a $(1, m+1)$ -block encoding of $(f_{even}(A) + f_{odd}(A))/2$. In other words, we obtain a circuit $U_f \in BE_{2,m+1}(f(A))$. note that unlike the case of block encoding of $P_{Re}(A)$, we lose a subnomalization factor of 2 here.

Following the same principle, if $f(x) = g(x) + \iota h(x) \in \mathbb{C}[x]$ is a given complex polynomial, and $g, h \in \mathbb{R}[x]$ do not have a definite parity, we can construct $U_{g(A)} \in BE_{2,m+1}(g(A))$, $U_{h(A)} \in BE_{2,m+1}(h(A))$. Then applying another layer of LCU, we obtain $U_{f(A)} \in BE_{4,m+3}(f(A))$.

On the other hand, if the real and imaginary parts g, h have definite parity, then $U_{g(A)} \in BE_{1,m+1}(g(A))$, $U_{h(A)} \in BE_{1,m+1}(h(A))$. Applying LCU, we obtain $U_{f(A)} \in BE_{2,m+1}(f(A))$.

Chapter 29

Quantum Signal Processing

29.1 Introduction

Quantum signal processing (QSP) describes a method for non linear transformations of a signal parameter encoded in a single-qubit gate, using a structured sequence that interleaves the “signal gate” with fixed parameterized “modulation” gates. The technique was originally motivated by the desire to characterize pulse sequences used in nuclear magnetic resonance. Remarkably, it has been shown that there is a rich family of polynomial transformations that are in one-to-one correspondence with appropriate modulation sequences, moreover given such a polynomial one can efficiently compute the corresponding modulation parameters.

Even more remarkably, this analysis holds not just for single-qubit “signal gates” but can be extended for multi-qubit operators that act like single-qubit rotations when restricted to the appropriate two-dimensional subspaces. This insight enables the implementation of block-encoding of polynomials of Hermitian/normal matrices when used in conjunction with qubitization. The two step process of qubitization + QSP can be unified and generalized through quantum singular value transformation (QSVT).

29.1.1 Overview

We follow the “Wx” convention of QSP. We define single-qubit operator

$$W(x) = \begin{pmatrix} x & \iota\sqrt{1-x^2} \\ \iota\sqrt{1-x^2} & x \end{pmatrix} = e^{\iota\arccos(x)X}$$

which is a single-qubit X rotations. We can verify that

$$\begin{aligned} W(x)^2 &= \begin{pmatrix} 2x^2 - 1 & \cdot \\ \cdot & \cdot \end{pmatrix} \\ W(x)^3 &= \begin{pmatrix} 4x^3 - 3x & \cdot \\ \cdot & \cdot \end{pmatrix} \\ &\vdots \\ W(x)^n &= \begin{pmatrix} T_n(x) & \cdot \\ \cdot & \cdot \end{pmatrix} \end{aligned}$$

where $T_n(x)$ is the n -th Chebyshev polynomial of the first kind, showcasing that even a simple sequence of the signal unitaries can implement a rich family of the signal x .

More complex behavior is obtained by interleaving $W(x)$ with parameterized single-qubit rotations $e^{i\phi_j Z}$. We define a QSP sequence.

$$U_{QSP}(\Phi) = e^{i\phi_0 Z} \prod_{j=1}^d W(x) e^{i\phi_j Z}$$

where Φ denotes the vector of angles $(\phi_0, \phi_1, \dots, \phi_d)$. The QSP sequences implements the following unitary

$$U_{QSP}(\Phi) = \begin{pmatrix} P(x) & iQ(x)\sqrt{1-x^2} \\ iQ^*(x)\sqrt{1-x^2} & P^*(x) \end{pmatrix}$$

where $P(x), Q(x)$ are complex polynomials obeying a number of constraints, and $P^*(x), Q^*(x)$ denote their complex conjugates.

In terms of implementing matrix polynomials of hermitian matrices, quantum eigenvalue transform provides a much simpler methods than based on the LCU and qubitization (i.e., linear combination of Chebyshev polynomials). The simplification is clear both in terms of the number of ancilla qubits and of the circuit architecture. However, it is not clear so far for which polynomials (either complex polynomial $P \in \mathbb{C}[x]$ or real polynomial $P_{Re} \in \mathbb{R}[x]$) we can apply the QET technique, and how to obtain the phase factors. Quantum signal processing (QSP) provides a complete answer to this question.

Due to qubitization, all these questions can be answered in the context of $SU(2)$ matrices. QSP is the theory of QET for $SU(2)$ matrices, otr the unitary representation of a scalar (real or complex) polynomial $P(x)$. Let $A = x \in [-1, 1]$, be a scalar with one-qubit Hermitian block encoding

$$U_A(x) = \begin{pmatrix} x & \sqrt{1-x^2} \\ \sqrt{1-x^2} & -x \end{pmatrix}$$

Then

$$O(x) = U_A(x)Z = \begin{pmatrix} x & -\sqrt{1-x^2} \\ \sqrt{1-x^2} & x \end{pmatrix}$$

is a rotation matrix.

The QSP representation takes the following form

$$U(x) = e^{\iota\phi_0 Z} O(x) e^{\iota\phi_1 Z} O(x) \dots e^{\iota\phi_{d-1} Z} O(x) e^{\iota\phi_d Z}$$

By setting $\phi_0 = \dots = \phi_d = 0$, we immediately obtain the block encoding of the Chebyshev polynomial $T_d(x)$. The representation power of this formulation is characterized in the following theorem. In the following discussion, even functions have parity 0 and odd functions have parity 1.

Theorem 29.1.1. (Quantum Signal Processing) *There exists a set of phase factors $\Phi = (\phi_0, \dots, \phi_d) \in \mathbb{R}^{d+1}$ such that*

$$U_\Phi(x) = e^{\iota\phi_0 Z} \prod_{j=1}^d [O(x) e^{\iota\phi_j Z}] = \begin{pmatrix} P(x) & -Q(x)\sqrt{1-x^2} \\ Q^*(x)\sqrt{1-x^2} & P^*(x) \end{pmatrix}$$

if and only if $P, Q \in \mathbb{C}[x]$ satisfy

1. $\deg(P) \leq d, \deg(Q) \leq d-1$
2. P has parity d mod 2 and Q has parity $d-1$ mod 2, and
3. $|P(x)|^2 + (1-x^2)|Q(x)|^2 = 1 \forall x \in [-1, 1]$.

Here $\deg(Q) = -1$ means $Q = 0$.

Proof. Since both $e^{\iota\phi Z}$ and $O(x)$ are unitary, the matrix $U_\Phi(x)$ is always a unitary matrix, which immediately implies the condition (3). Below we only need to verify the conditions (1), (2).

When $d = 0$, $U_\Phi(x) = e^{\iota\phi_0 Z}$, which gives $P(x) = e^{\iota\phi_0}$ and $Q = 0$ satisfying all three conditions. For induction, suppose $U_{(\phi_0, \dots, \phi_{d-1})}(x)$ takes the form as given in the previous equation with degree $d-1$, then for any $\phi \in \mathbb{R}$, we have

$$\begin{aligned} U_{(\phi_0, \dots, \phi_{d-1}, \phi)}(x) &= \begin{pmatrix} P(x) & -Q(x)\sqrt{1-x^2} \\ Q^*(x)\sqrt{1-x^2} & P^*(x) \end{pmatrix} \begin{pmatrix} x & -\sqrt{1-x^2} \\ \sqrt{1-x^2} & x \end{pmatrix} \begin{pmatrix} e^{\iota\phi} & 0 \\ 0 & e^{-\iota\phi} \end{pmatrix} \\ &= \begin{pmatrix} xP(x) - (1-x^2)Q(x) & -\sqrt{1-x^2}(P(x) + xQ(x)) \\ -\sqrt{1-x^2}(P^*(x) + xQ^*(x)) & xP^*(x) - (1-x^2)Q^*(x) \end{pmatrix} \begin{pmatrix} e^{\iota\phi} & 0 \\ 0 & e^{-\iota\phi} \end{pmatrix} \\ &= \begin{pmatrix} e^{\iota\phi}(xP(x) - (1-x^2)Q(x)) & e^{-\iota\phi}(-\sqrt{1-x^2}(P(x) + xQ(x))) \\ e^{\iota\phi}((-xP(x) + (1-x^2)Q(x)) & e^{\iota\phi}(xP^*(x) - (1-x^2)Q^*(x)) \end{pmatrix} \end{aligned}$$

Therefore $U_{(\phi_0, \dots, \phi_{d-1}, \phi)}(x)$ satisfies the conditions (1),(2).

For the condition (1), note that the new $P'(x)$ can be written as:

$$P'(x) = e^{\iota\phi}(xP(x) - (x - x^2)Q(x))$$

Thus, if $\deg(P(x)) \leq d - 1 \implies \deg(P'(x)) \leq d$ because of the x multiplied. Similarly,

$$\begin{aligned} -\sqrt{1-x^2}Q'(x) &= e^{-\iota\phi}(-\sqrt{1-x^2}(P(x) + xQ(x))) \\ &= -\sqrt{1-x^2}\left(e^{-\iota\phi}\left(P(x) + \frac{x}{\sqrt{1-x^2}}Q(x)\right)\right) \\ Q'(x) &= e^{-\iota\phi}\left(P(x) + \frac{x}{\sqrt{1-x^2}}Q(x)\right) \end{aligned}$$

Thus, clearly if $\deg(Q(x)) \leq d - 2 \implies \deg(Q'(x)) \leq d - 1$.

we can also prove the condition (2) by taking different cases that let d be even then proving that $P'(x)$ is even and $Q'(x)$ is odd function. Similarly let d be odd, we can then prove that $P'(x)$ is an odd function and $Q'(x)$ is an even function.

Because it is an if and only if condition we now prove the other way around. When $d = 0$ the only possibility is $P(x) = e^{\iota\phi_0}$ and $Q = 0$, which satisfies the equation.

For $d > 0$, when d is even we may still have $\deg(P) = 0$ i.e., $P(x) = e^{\iota\phi_0}$. In this case, note that

$$O^{-1}(x) = O^\dagger(x) = \begin{pmatrix} x & \sqrt{1-x^2} \\ -\sqrt{1-x^2} & x \end{pmatrix} = e^{-\iota\frac{\pi}{2}Z}O(x) + e^{\iota\frac{\pi}{2}Z}$$

we may set $\phi_j = (-1)^j\frac{\pi}{2}$, $j = 1, \dots, d$, and

$$e^{\iota\phi_0Z} \prod_{j=1}^d [O(x)e^{\iota\phi_jZ}] = e^{\iota\phi_0Z}(O(x)O^\dagger(x))^{\frac{d}{2}} = e^{\iota\phi_0Z}$$

Thus, the statement holds.

Now given P, Q satisfying the conditions (1) – (3), with $\deg(P) = l > 0$ and $l \equiv d \pmod{2}$ (meaning that $l - d = 2k$, thus $l - d$ is even, which means if l is odd then d is odd and l is even than d is even). Then $\deg(|P(x)|^2) = 2l > 0$, and according to the condition (3) we must have $\deg(Q) = l - 1$. Let P, Q be expanded as

$$P(x) = \sum_{k=0}^l -k = 0^l \alpha_k x^k, \quad Q(x) = \sum_{k=0}^{l-1} \beta_k x^k$$

then the leading term of $|P(x)|^2 + (1 - x^2)|Q(x)|^2 = 1$ is

$$|\alpha_l|^2 x^{2l} - x^2 |\beta_{l-1}|^2 x^{2l-2} = (|\alpha_l|^2 - |\beta_{l-1}|^2) x^{2l} = 0$$

which implies $|\alpha_l| = |\beta_{l-1}|$.

For any $\phi \in \mathbb{R}$, we have

$$\begin{aligned} & \begin{pmatrix} P(x) & -Q(x)\sqrt{1-x^2} \\ Q^*(x)\sqrt{1-x^2} & P^*(x) \end{pmatrix} e^{-\imath\phi Z} O^\dagger(x) \\ &= \begin{pmatrix} P(x) & -Q(x)\sqrt{1-x^2} \\ Q^*(x)\sqrt{1-x^2} & P^*(x) \end{pmatrix} \begin{pmatrix} e^{-\imath\phi} & 0 \\ 0 & e^{\imath\phi} \end{pmatrix} \begin{pmatrix} x & \sqrt{1-x^2} \\ -\sqrt{1-x^2} & x \end{pmatrix} \\ &= \begin{pmatrix} e^{-\imath\phi} x P(x) + (1-x^2) Q(x) e^{\imath\phi} & -\sqrt{1-x^2} (e^{-\imath\phi} P(x) + x Q(x) e^{\imath\phi}) \\ \sqrt{1-x^2} (-e^{-\imath\phi} P^*(x) + x Q^*(x) e^{-\imath\phi}) & e^{\imath\phi} x P^*(x) + (1-x^2) Q^*(x) e^{-\imath\phi} \end{pmatrix} \\ &= \begin{pmatrix} \tilde{P}(x) & -\tilde{Q}(x)\sqrt{1-x^2} \\ \tilde{Q}^*(x)\sqrt{1-x^2} & \tilde{P}^*(x) \end{pmatrix} \end{aligned}$$

It may appear that $\deg(\tilde{P}) = l+1$. However by properly choosing ϕ we may obtain $\deg(\tilde{P}) = l-1$. let $e^{2\imath\phi} = \alpha_l/\beta_{l-1}$. Then the coefficient of the x^{l+1} term in \tilde{P} is

$$e^{-\imath\phi} \alpha_l - e^{\imath\phi} \beta_{l-1} = \frac{\alpha_l + e^{2\imath\phi} \beta_{l-1}}{e^{\imath\phi}} = 0$$

Similarly, the coefficient of the x^l term in \tilde{Q} is

$$-e^{\imath\phi} \alpha_l + e^{\imath\phi} \beta_{l-1} = 0$$

The coefficient of the x^l term in \tilde{P} , and the coefficient of the x^{l-1} term in \tilde{Q} are both 0 by the parity condition. So we have

1. $\deg(\tilde{P}(x)) \leq l-1 \leq d-1$, $\deg(Q) \leq l-2 \leq d-2$.
2. \tilde{P} has parity $d-1 \pmod{2}$ and \tilde{Q} has parity $d-2 \pmod{2}$, and
3. $|\tilde{P}(x)|^2 + (1-x^2)|\tilde{Q}(x)|^2 = 1$, $\forall x \in [-1, 1]$.

Here the condition (3) is automatically satisfies due to unitary. The induction follows until $l=0$, and applying the argument to represent the remaining constant phase factor if needed. \square

Remark. (**W convention of QSP**) It is stated slightly differently as

$$U_{\Phi^W}(x) = e^{\iota\phi_0^W Z} \prod_{j=1}^d [W(x)e^{\iota\phi_j^W Z}] = \begin{pmatrix} P(x) & \iota Q(x)\sqrt{1-x^2} \\ \iota Q^*(x)\sqrt{1-x^2} & P^*(x) \end{pmatrix}$$

where

$$W(x) = e^{\iota \arccos(x) X} = \begin{pmatrix} x & \iota\sqrt{1-x^2} \\ \iota\sqrt{1-x^2} & x \end{pmatrix}$$

This will be referred to as the W-convention. Correspondingly the previous equation will be referred to as the O-convention. The two conventions can be easily converted into one another, due to the relation

$$W(x) = e^{-\iota\frac{\pi}{4}Z} O(x) e^{\iota\frac{\pi}{4}Z}$$

Correspondingly the relation between the phase angles using the O and W representation are related according to

$$\phi_j = \begin{cases} \phi_0^W - \frac{\pi}{4}, & \text{if } j = 0 \\ \phi_j^W, & \text{if } j = 1, \dots, d-1 \\ \phi_d^W + \frac{\pi}{4}, & \text{if } j = d \end{cases}$$

On the other hand, note that for any $\theta \in \mathbb{R}$, $U_\Phi(x)$ and $e^{\iota\theta Z} U - \Phi(x)e^{-\iota\theta Z}$ both block encodes $P(x)$. Therefore without loss of generality we may as well take

$$\Phi = \Phi^W$$

In many applications, we are only interested in $P \in \mathbb{C}[x]$, and $Q \in \mathbb{C}[x]$ is not provided a priori. The theorem in [?] states that under certain conditions P , the polynomial Q can always be constructed. We omit the details here.

29.2 QSP for real polynomials

Note that the normalization condition (3) in theorem imposes very strong constraints on the coefficients of $P, Q \in \mathbb{C}[x]$. If we are only interested in QSP for real polynomials, the condition can be significantly relaxed.

Theorem 29.2.1. (*Quantum signal processing for real polynomials*) Given a real polynomial $P_{Re}(x) \in \mathbb{R}[x]$, and $\deg(P_{Re}) = d > 0$ satisfying

1. P_{Re} has parity $d \bmod 2$.

2. $|P_{Re}| \leq 1 \forall x \in [-1, 1]$

then there exists polynomials $P(x), Q(x) \in \mathbb{C}[x]$ with $Re(P) = P_{Re}$ and a set of phase factors $\Phi = (\phi_0, \dots, \phi_d) \in \mathbb{R}^{d+1}$ such that the QSP representation holds.

29.3 Dominant resource cost - gates and qubits

A QSP circuit that implements a degree d polynomials in the signal parameter requires d uses of $W(x)$ and $d + 1$ fixed angle Z rotations. There are efficient classical algorithms to determine the angles for a given target polynomials, either using high-precision arithmetic with $\approx d \log(d)$ bits of precision (or more - though this can be mitigated using heuristic techniques) or in some regimes using more efficient optimization-based algorithms. Although these procedures are efficient in theory, in practice it may still be non-trivial to find the angles. Nevertheless, researches reportedly computed angle sequences corresponding to various degree $\mathcal{O}(10^4)$ polynomials.

Important Note

As discussed above, not all polynomials can be implemented by a QSP sequence. Implementable polynomials must obey a number of constraints which can be somewhat restrictive. For the standard QSP circuit $U_{QSP}(\Phi)$ given above, the achievable polynomial pairs $P(x), Q(x) \in \mathbb{C}$ can be characterized by the following three conditions:

- $\text{Deg}(P) \leq d, \quad \text{Deg}(Q) \leq d - 1$
- $\text{Parity}(P) = \text{Parity}(d), \quad \text{Parity}(Q) = \text{Parity}(d - 1)$
- $\forall x \in [-1, 1] : |P(x)|^2 + (1 - x^2)|Q(x)|^2 = 1$ (required to be Unitary)

This last requirement can be particularly limiting. A useful way to circumvent this for real functions is to encode the polynomial in the matrix element $\langle + | U_{QSP}(\Phi) | + \rangle$ rather than in $\langle 0 | U_{QSP}(\Phi) | 0 \rangle$, where $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$. This matrix element evaluates to

$$\langle + | U_{QSP}(\Phi) | + \rangle = \text{Re}[P(x)] + i\sqrt{1 - x^2}\text{Re}[Q(x)]$$

Given a real target polynomial $f(x)$ with parity equal to $\text{Parity}(d)$, we can guarantee that the matrix element evaluates to $f(x)$ by choosing $\text{Re}[P(x)] = f(x)$ and $\text{Re}[Q(x)] = 0$. The third condition above then reduces to $1 - f(x)^2 = |\text{Im}[P(x)]|^2 + (q - x^2)|\text{Im}[Q(x)]|^2$. There exist choices for $\text{Im}[P(x)]$ and $\text{Im}[Q(x)]$ that satisfy this identity as well as the first two conditions above, provided $f(x) \leq 1 \forall x \in [-1, 1]$. In summary, we may implement any real polynomial $f(x)$ satisfying the requirements.

- $\text{Deg}(f) = d$
- $\text{Parity}(f) = \text{Parity}(d)$
- $\forall x \in [-1, 1] : |f(x)| \leq 1$

There are several related conventions considered in the literature for the explicit form of the single qubit operators used in QSP. One common form that links closely to qubitization and QSFT is the reflection conventions, which replaces $W(x)$ by the reflection

$$R(x) = \begin{bmatrix} x & \sqrt{1 - x^2} \\ \sqrt{1 - x^2} & -x \end{bmatrix}$$

and adjusts the parameters $\{\phi_j\}$ accordingly.

Some example uses cases include:

- Functions of Hermitian/normal matrices, in conjunction with qubitization, including for Hamiltonian simulation.
- Functions of general matrices via quantum singular value transformation (QSVT).
- QSP applied to beyond-Heisenberg limit calibration of two-qubit gates in a superconducting system.

Chapter 30

Quantum Singular Value Transformation

In previous chapters, we have found that using qubitization, we can effectively block encode the Chebyshev matrix polynomial $T_k(A)$ for a Hermitian matrix A . Combined with LCU, we can construct a block encoding of any matrix polynomial of A . The process is greatly simplified using QSP and QET, which allows the implementation of a general class of matrix functions for Hermitian matrices.

In this section, we generalize the results of qubitization and QET to general non-Hermitian matrix. This is called the quantum singular value transformation (QSVT). Throughout the chapter we assume $A \in \mathbb{C}^{N \times N}$ is a square matrix. QSVT is applicable to non-square matrices as well, and we will omit the discussions here.

30.1 Generalized Matrix functions

For a square matrix $A \in \mathbb{C}^{N \times N}$, where for simplicity we assume $N = 2^n$ for some positive integer n , the singular value decomposition (SVD) of the normalized matrix A can be written as

$$A = W\Sigma V^\dagger$$

or equivalently

$$A|v_i\rangle = \sigma_i|w_i\rangle, \quad A^\dagger|w_i\rangle = \sigma_i|v_i\rangle, \quad i \in [N]$$

We may apply a function $f(\cdot)$ on its singular values and define the generalized matrix function as below.

Definition 30.1.1. (Generalized matrix function) Given $A \in \mathbb{C}^{N \times N}$ with singular value decomposition as given above, and let $f : \mathbb{R} \rightarrow \mathbb{C}$ be a scalar function

such that $f(\sigma_i)$ is defined for all $i \in [N]$. The generalized matrix function is defined as

$$f^o(A)$$

Part III

Quantum Complexity Theory

Chapter 31

Computational Complexity Theory

31.1 Introduction

The modern incarnation of computer science was announced by the great mathematician Alan Turing in a remarkable 1936 paper. Turing developed in detail an abstract notion of what we would now call a programmable computer, a model for computation now known as *Turing machine*. Turing showed that there is a *Universal Turing Machine* that can be used to simulate any other *Turing Machine*. He claimed that Universal Turing Machine can be used to simulate any other Turing machine. The Universal Turing Machine completely captures what it means to perform a task by algorithmic means. That is, if an algorithm can be performed on any given piece of hardware (say, a modern computer), then there is an equivalent algorithm for a Universal Turing Machine which performs exactly the same task as the algorithm running on the personal computer. This assertion is known as **Church-Turing Hypothesis**, asserts the equivalence between the physical concept of what class of algorithms can be performed on some physical device with rigorous mathematical concept of a Turing Machine.

Not long after Turing's paper, John von Neumann developed a simple theoretical model for how to put together in a practical fashion all the components necessary for a computer to be fully capable as a Universal Turing Machine. Hardware development took off when John Bradeen, Walter Trattain, and Will Shockley developed the transistor in 1947. Computer hardware has grown in power at an amazing speed, so much so that their growth was codified by Gordon Moore in 1965 in what has come to be known as Moore's law which states that computer power will double for constant cost roughly once every two years.

Moore's law was approximately held true in the decades since the 1960s. Most

observers expect that this dream run will end in some time during the first two decades of the twenty-first century. Conventional approaches to the fabrication of computer technology are beginning to run up against fundamental difficulties of size. Quantum effects are beginning to interfere in the functioning of electronic devices as they are made smaller and smaller.

One possible solution to the problem posed by the eventual failure of Moore's law is to move to a different computing paradigm. One such paradigm is provided by the theory of quantum computation., which is based on the idea of using quantum mechanics to perform computations, instead of classical physics. It turns out that while an ordinary computer can be used to simulate a quantum computer it appears to be impossible to perform the simulation in an efficient fashion. Thus quantum computers offer an essential speed advantage over classical computers. This speed advantage is so significant that many researchers believe that no conceivable amount of progress in classical computation would be able to overcome the gap between the power of a classical computer and the power of a quantum computer.

31.1.1 Efficient vs Inefficient

What do we mean by ‘efficient’ versus ‘inefficient’ simulations of a quantum computer? The idea of efficient and inefficient algorithms was made mathematically precise by the field of computational complexity. Roughly speaking, an efficient algorithm is one which runs in time polynomial in the size of the problem solved. In contrast, an inefficient algorithm requires super-polynomial (typically exponential) time.

Turing machine model was at least as powerful as any other model of computation, in the sense that a problem which can be solved efficiently in some model of computation could also be solved efficiently in the Turing machine model, by using the Turing machine to simulate the other model of computation. This leads us to the Church-Turing Thesis

Any algorithmic process can be simulated efficiently using a Turing machine

The key strengthening in the strong Church-Turing Thesis is the word efficiently. If the strong Church-Turing thesis is correct, then it implies that no matter what type of machine we use to perform our algorithms , that machine can be simulated efficiently using a standard Turing machine. This implies that for the purposes of analyzing whether a given computational task can be accomplished efficiently, we may restrict ourselves to the analysis of the Turing model of computation.

31.1.2 The Challenges

One class of challenges to the strong Church-Turing thesis comes from the field of analog computation. In the years since Turing, many different teams of researchers have noticed that certain types of analog computers can efficiently solve problems believed to have no efficient solution on a Turing Machine. At first glance these analog computers appear to violate the strong form of the Church-Turing thesis. Unfortunately for analog computation, it turns out that when realistic assumptions about the presence of noise in analog computers are made, their power disappears in all known instances; they cannot efficiently solve problem which are not efficiently solvable on a Turing machine. This lesson - that the effects of realistic noise must be taken into account in evaluating the efficiency of a computational model - was one of the great early challenges of quantum computation and quantum information, a challenge successfully met by the development of a theory of quantum error-correcting codes and fault-tolerant quantum computation. Thus, unlike analog computation, quantum computation can in principle tolerate a finite amount of noise and still retain its computational advantages.

The first major challenge to the Strong Church-Turing thesis arose in the mid 1970s when Robert Solovay and Volker Strassen showed that it is possible to test whether an integer is prime or composite using a randomized algorithm. That is the Solvay-Strassen test for primality used randomness as an essential part of the algorithm. The algorithm did not determine whether a given integer was prime or composite with certainty. By repeating the Solovay-Strassen test a few times it is possible to determine with near certainty whether a number is prime or composite. The Solovay-Strassen test was of special significance at the time it was proposed as no deterministic test for primality was then known, nor is one known at the time of this writing. Thus, it seemed as though computer with access to random number generator would be able to efficiently perform computational tasks with no efficient solution on a conventional deterministic Turing machine. This discovery inspired a search for other randomized algorithms which has paid off handsomely, with the field blossoming into a thriving area of research.

Randomized algorithms pose a challenge to the strong Church-Turing Thesis, suggesting that there are efficiently solvable problems which, nevertheless, cannot be efficiently solved on a deterministic Turing machine. This challenge appears to be easily resolved by a simple modification of the strong Church-Turing thesis:

Any algorithmic process can be simulated efficiently using a probabilistic Turing machine

This ad hoc modification of the strong Church-Turing thesis should leave you

feeling rather queasy. Might it not turn out at some later date that yet another model of computation allows one to efficiently solve problems that are not efficiently solvable within Turing's model of computation? Is there any way we can find a single model of computation which is guaranteed to be able to efficiently simulate any other model of computation?

Motivated by this question, in 1985 David Deutsch asked whether the laws of physics could be used to derive an even stronger version of the Church-Turing thesis. Instead of ad hoc hypotheses, Deutsch looked to physical theory to provide a foundation for the Church-Turing thesis that would be as secure as the status of that physical theory. In particular, Deutsch attempted to define a computational device that would be capable of effectively simulating an arbitrary physical system. These devices, quantum analogues of the machines defined forty-nine years earlier by Turing, led ultimately to the modern conception of a quantum computer.

At the time of writing it is not clear whether Deutsch's notion of a Universal Quantum Computer is sufficient to efficiently simulate an arbitrary Physical system. Proving or refuting this conjecture is one of the great problems of the field of quantum computation and quantum information. It is possible, for example, that some effect of quantum field theory or an even more esoteric effect based in string theory, quantum gravity or some other physical theory may take us beyond Deutsch's Universal Quantum computer, giving us a still more powerful model for computation. At this stage, we simply don't know.

What Deutsch's model of a quantum computer did enable was a challenge to the Strong form of the Church-Turing thesis. Deutsch asked whether it is possible for a quantum computer to efficiently solve computational problems which have no efficient solution on a classical computer, even a probabilistic Turing machine. He then constructed a simple example suggesting that, indeed, quantum computers might have computational powers exceeding those of classical computers.

This remarkable first step taken by Deutsch was improved in the subsequent decade by many people, culminating in Peter Shor's 1994 demonstration that two enormously important problems - the problem of finding the prime factors of an integer, and the so-called 'discrete logarithm' problem - could be solved efficiently on a quantum computer. This attracted widespread interest because these two problems were and still are widely believed to have no efficient solution on a classical computer. Shor's result are a powerful indication that quantum computers are more powerful than Turing machines, even probabilistic Turing machines. Further evidence for the power of quantum computers came in 1995 when Lov Grover showed that another important problem - the problem of conducting a search through some unstructured search space - could also be sped up on a quantum computer. While Grover's al-

gorithm did not prove as spectacular speed up as Shor's algorithms, the widespread applicability has excited considerable interest in grover's algorithm.

At about the same time as Shor's and Grover's algorithm were discovered, many people were developing an idea Richard Feynman had suggest in 1982. Feynman had pointed out that there seemed to be essential difficulties in simulating quantum mechanical systems on classical computers, and suggested that building computers based on the principles of quantum mechanics would allow us to avoid those difficulties. In the 1990s several teams of researchers began fleshing this idea out, showing that it is indeed possible to use quantum computers to efficiently simulate systems that have no known efficient simulation on a classical computers. It is likely that one of the major applications of quantum computers in the future will be performing simulations of quantum mechanical systems too difficult to simulate on a classical computer, a problem with profound scientific and technological implications.

31.1.3 What other problems?

What other problems can quantum computers solve more quickly than classical computer? The short answer is that we don't know. Coming up with good quantum algorithms seems to be hard. It is because algorithm design for quantum computers is hard because designers face two difficult problems not faced in the constructing of algorithms for classical computer. First, our human intuition is rooted in the classical world. If we use that intuition as an aid to the construction of algorithms, then the algorithmic ideas we come up with will be classical ideas. To design good quantum algorithms one must turn off one's classical intuition for at least part of the design process, using truly quantum effects to achieve the desired algorithmic end. Second, to be truly interesting it is not enough to design an algorithm that is merely quantum mechanical. The algorithm must be better than any existing classical algorithm. Thus! it is possible that one may find an algorithm which makes use of truly quantum aspects of quantum mechanics, that is nevertheless not of widespread interest because classical algorithms with comparable performance characteristics exist. The combination of these two problems makes the construction of new quantum algorithms a challenging problem for the future.

Even more broadly, we can ask if there are any generalizations we can make about the power of quantum computers versus classical computers. What is it that makes quantum computers more powerful than classical computers. What is it that makes quantum computers more powerful than classical computers. Assuming that this is indeed the case? What class of problems can be solved efficiently on a quantum computer, and how does that class compare to the class of problems that can be

solved efficiently on a classical computer? One of the most exciting things about quantum computation and quantum information is how little is known about the answers to these questions! It is a great challenge for the future to understand these questions better.

31.2 The power of quantum computation

How powerful are quantum computers? What gives them their power? Nobody yet knows the answers to these questions, despite the suspicions fostered by examples such as factoring, which strongly suggests that quantum computers are more powerful than classical computers. It is still possible that quantum computers are no more powerful than classical computers, in the sense that any problem which can be efficiently solved on a quantum computer can also be efficiently solved on a classical computer. On the other hand, it may eventually be proved that quantum computers are much more powerful than classical computers.

Definition 31.2.1. (Algorithm): An algorithm is a precise recipe for performing some task. For example, adding two numbers

The fundamental question we are trying to address in the study of algorithms is: what resources are required to perform a given computational task. This question splits up into two parts:

1. What computational tasks are possible, preferably by giving explicit algorithms for solving specific problems. For example, algorithms that can quickly sort a list of numbers into ascending order.
2. To demonstrate limitations on what computational tasks may be accomplished. For example, lower bounds can be given for the number of operations that must be performed by any algorithm which sorts a list of numbers into ascending order.

Ideally, these two tasks - the finding of algorithms for solving computational problems, and proving limitations on our ability to solve computational problems, would dovetail perfectly. In practice, a significant gap often exists between the best techniques known for solving a computational problems, and the most stringent limitations known on the solution.

31.2.1 Why study classical computer science?

1. Classical computer science provides a cast body of concepts and techniques which may be reused to great effect in quantum computation and quantum information. Many of the triumphs of the quantum computation and quantum information have come by combining existing ideas from computer science with novel ideas from quantum mechanics. For example, some of the fast algorithms for quantum computers are based upon the Fourier transform, a powerful tool utilized by many classical algorithms. Once it was realized that quantum computers can perform a type of Fourier transform much more quickly than classical computer this enabled the development of many important quantum algorithms.
2. Computer scientists have expended great effort understanding what resources are required to perform a given computational tasks on a classical computer. These results can be used as the basis for a comparison with quantum computation and quantum information. For example, the problem of finding prime factors of a given number. On a classical computer this problem is believed to have no ‘efficient’ solution, here ‘efficient’ has a meaning to be explained later. An efficient solution to this problem is known for quantum computers. Thus, for the task of finding prime factors there appears to be a gap between what’s possible on a classical computer and what is possible on a quantum computer. Thus a gap may exist for a wider class of computation problem than merely finding of prime factors. By studying this specific problem further, it may be possible to discern features of the problem which make it more tractable on a quantum computer than on a classical computer, and then act on these insights to find interesting quantum algorithms for the solution to other problems.
3. These is learning to think like a computer scientist. Computer scientists think in a rather different style than does a physicist. Thus for deep understanding of quantum computation and quantum information one must learn to think like a computer scientist at least some of the time, they must instinctively known what problems, what techniques and mot importantly what problems are of greatest interest.

We now take a brief look at what is known about the power of quantum computation.

31.2.2 Computational Complexity Theory

It is the subject of classifying the difficulty of various computational problems, both classical and quantum, and to understand the power of quantum computers we will first examine some general ideas from computational complexity. The most basic idea is that of a complexity class. A complexity class can be thought of as a collection of computational problems, all of which share some common feature with respect to the computational resources needed to solve those problems.

Two of the most important complexity classes go by the names P and NP . Roughly speaking, P is the class of computational problems that can be solved quickly on a classical computer. NP is the class of problems which have solutions which can be quickly checked on a classical computer. To understand the distinction between P and NP , consider the problem of finding the prime factors of an integer, n . So far as is known there is no fast way of solving this problem on a classical computer, which suggests that the problem is not in P . On the other hand, if somebody tells you that some number p is a factor of n , then we can quickly tell that this is correct by dividing p into n , so factoring is a problem in NP .

It is clear that P is a subset of NP , since the ability to solve a problem implies the ability to check potential solutions. What is not clear is whether or not there are problems in NP that are not in P . Perhaps the most important unsolved problem in theoretical science is to determine whether these two classes are different:

$$P \neq NP$$

Most researchers believe that NP contains problems that are not in P . In particular there is an important subclass of the NP problems, the NP -complete problems, that are of especial importance for two reasons. First, there are thousands of problems, many highly important, that are known to be NP -complete. Second, any given NP -complete problem is in some sense ‘at least as hard’ as all other problems in NP . More precisely, an algorithm to solve a specific NP -complete problem can be adapted to solve any other problem in NP , with a small overhead. In particular, if $P \neq NP$, then it will follow that no NP -complete problem can be efficiently solved on a classical computer.

It is not known whether quantum computers can be used to quickly solve all the problems in NP , despite the fact that they can be used to solve problems - like factoring which are believed by many people to be in NP but not in P . (Note that factoring is not known to be NP -complete problem, otherwise we would already know how to efficiently solve all problems in NP using quantum computers). It would certainly be very exciting if it were possible to solve all the problems in NP

efficiently on quantum computer. There is a very interesting negative result known in this direction which rules out using a simple variant of quantum parallelism to solve all problems in NP . Specifically, one approach to the problem of solving problems in NP on a quantum computer is to try to use some form of quantum parallelism to search in parallel through all possible solutions to the problem. We will later show that no approach based upon such a search-based methodology can yield an efficient solution to all the problems in NP . While it is disappointing that this approach fails, it does not rule that some deeper structure exists in NP that will allow them all to be solved quickly using a quantum computer.

P and NP are just two of a plethora of complexity classes that have been defined. Another important complexity class is $PSPACE$. Roughly speaking, $PSPACE$ consists of those problems which can be solved using resources which are few in spatial size (that is, the computer is ‘small’), but not necessarily in time (‘long’ computations are fine). $PSPACE$ is believed to be strictly greater than both P and NP although, again, this has never been proved. Finally, the complexity class BPP is the class of problems that can be solved using randomized algorithms in polynomial time, if a bounded probability of error (say $1/4$) is allowed in the solution to the problem. BPP is widely regarded as being, even more so than P , the class of problems which should be considered efficiently soluble on a classical computer. We have elected to concentrate here on P rather than BPP because P has been studied in more depth, however many similar ideas and conclusions arise in connection with BPP .

What of quantum complexity classes? We can define BQP to be the class of all computational problems which can be solved efficiently on a quantum computer, where a bounded probability of error is allowed. (strictly speaking this makes BQP more analogous to the classical complexity class BPP than to P , however we will ignore this subtlety for the purpose of the present discussion, and treat it as the analogue of P .) Exactly where BQP fits with respect to $P, NP, PSPACE$ is yet unknown. Therefore, BQP lies somewhere between P and $PSPACE$. An important implication is that if it is proved that quantum computers are strictly more powerful than classical computers, then it will follow that P is not equal to $PSPACE$. Proving this latter result has been attempted without success by many computer scientists, suggesting that it may be non-trivial to prove that quantum computers are more powerful than classical computers, despite much evidence in favor of this proposition.

It is clear that the theory of quantum computation poses interesting and significant challenges to the traditional notions of computation. What makes this an important challenge is that the theoretical model of quantum computation is be-

lieved to be experimentally realizable, because to the best of our knowledge - this theory is consistent with the way Nature works. If this were not so then quantum computation would be just another mathematical curiosity.

31.3 Models of Computation

The fundamental model for algorithms will be the *Turing machine*. This is an idealized computer, like a modern personal computer, but with a simpler set of basic instructions, and an idealized unbounded memory. The apparent simplicity of Turing machines is misleading; they are very powerful devices. They can be used to execute any algorithm whatsoever, even one running on an apparently much powerful computer.

What does it mean for an algorithm to perform some task. for example adding any two numbers, no matter how large those numbers are.

One learns Euclid's two thousand year old algorithm for finding greatest common divisor of two positive integers. In the late 1930s the fundamental notions of the modern theory of algorithms, and thus of computation, were introduced, by Alonzo Church, Alan Turing and others. This work arose in response to a profound challenge laid down by the great mathematician David Hilbert in the early part of the twentieth century. Hilbert asked whether or not there existed some algorithm which could be used, in principle, to solve all the problems of mathematics. Hilbert expect that the answer to this question, sometimes known as the *entscheidungsproblem* would be yes.

The answer to Hilbert's challenge turned out to be no: there is no algorithm to solve all mathematical problems. To prove this, Church and Turing had to solve the deep problem of capturing a mathematical definition what we mean when we use the intuitive concept of an algorithm. In doing s, they laid the foundations for the modern theory of algorithms, and consequently for the modern theory of computer science.

We now discuss the approach proposed by Turing, the Turing machines. Turing defined a class of machines known as Turing machines, in order to capture the notion of an algorithm to perform a computational task. We will then also discuss the circuit model of computation. Although these models of computation appear different ont he surface, it turns out that they are equivalent. Introducing different models of computation may yield different insights into the solutions of specific problems. Two or more ways of thinking about a concept are better than one.

31.3.1 Turing Machines

every two years. 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 31.3.1. 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 31.3.2. 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 31.3.3. 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 31.3.4. EXP (exponential time) Solvable by $\mathcal{O}(2^n)$ -size classical circuits.

The following containment among these complexity classes are known:

$$\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{EXP}$$

Chapter 32

Loading classical data

The end-to-end quantum applications covered in this document have classical inputs and classical outputs, in the sense that the problem is specified by some set of classical data, and the solution to the problem should be a different set of classical data. In some cases, the input data is relatively small, and loading it into the algorithm does not contribute significantly to the cost of the algorithm. In other cases for example, “big data” problems within the areas of machine learning and finance - the dominant costs, both for classical and quantum algorithms, can be related to how the algorithms load and manipulate this large quantity of input data. Consequently, the availability of quantum speedups for these problems if often dependent on the ability to quickly and coherently access this data. The true cost of this access is the source of significant subtlety in many end-to-end quantum algorithms.

32.1 Quantum random access memory

Quantum random access memory (QRAM) is a construction that enables coherent access to classical data, such that multiple different elements in a classical database can be read in superposition. The ability to access large, unstructured classical data sets in this way is crucial to the speedup of certain quantum algorithms (for example, quantum machine learning based on quantum linear algebra). QRAM is commonly invoked in such cases as a way to circumvent data-input bottlenecks, i.e., situations where loading data could limit the end-to-end runtime of an algorithm. It remains an open question, however, whether a large-scale QRAM will ever be practical, casting doubt on quantum speedups that rely on QRAM. Note that, while here we focus on the more common use case of loading classical data with QRAM, certain QRAM architectures can be adapted to also load quantum data.

Consider a length N , unstructured classical data vector x , and denote the i th entry as x_i . Let the number of bits of x_i be denoted by d and let $D = 2^d$. Given an input quantum state $|\psi\rangle = \sum_{i=0}^{N-1} \sum_{j=0}^{D-1} \alpha_{ij} |i\rangle_A |j\rangle_B$, QRAM is defined as a unitary operation Q with the action,

$$Q |\psi\rangle = Q \sum_{i=0}^{N-1} \sum_{j=0}^{D-1} \alpha_{ij} |i\rangle_A |j\rangle_B = \sum_{i=0}^{N-1} \sum_{j=0}^{D-1} \alpha_{ij} |i\rangle_A |j \oplus x_i\rangle_B$$

Here, A is a $\log_2(N)$ -qubit register, and B is a d -qubit register. Note that the unitary Q can also be understood as an oracle (or a black box) providing access to x , as $Q(\sum_i \alpha_i |i\rangle |0\rangle) = \sum_i \alpha_i |i\rangle |x_i\rangle$.

Let T_Q denote the time it takes to implement the operation Q , where T_Q can be measured in circuit depth, total gate cost, T gate cost, etc.., depending on the context. Algorithms that rely on QRAM to claim exponential speedups over their classical counterparts frequently assume $T_Q = \text{polylog}(N)$.

32.1.1 Resource cost

The QRAM operation Q can be implemented as a quantum circuit that uses $\mathcal{O}(N)$ ancillary qubits and $\mathcal{O}(N)$ gates. Assuming gates acting on disjoint qubits can be parallelized, the depth of the circuit is only $T_Q = \mathcal{O}(\log(N))$. Explicit circuits can be found in e.g. The number of ancillary qubits can be reduced at the price of increased circuit depth; circuit implementing Q can be constructed using $\mathcal{O}(N/M)$ ancillary qubits and depth $\mathcal{O}(M \log(N))$, where $M \in [1, N]$ see example

Part IV

Quantum Information Theory

Chapter 33

Introduction to Quantum Information

So far, this course has focused almost entirely on quantum algorithms. In the beginning of the course I described the model of quantum information that we have used up to this point. Specifically, the model describes states of qubits as unit vectors, and the allowed operations come from a fairly restricted set (unitary operations and measurements of a simple type). This description was sufficient for discussing quantum algorithms, so it has not been necessary to extend it until now. This is an alternate formulation using a tool known as the density operator or density matrix. This alternative formulation is mathematically equivalent to the state vector approach, but provides a much more convenient language for thinking about some commonly encountered scenarios in quantum mechanics.

To describe the model generally model of quantum information, some notation will be helpful. First, for any finite non empty set Σ , let $\mathbb{C}(\Sigma)$ denote the vector space of all column vectors indexed by Σ . Just as before, elements of such spaces are denoted by kets, e.g.e $|\psi\rangle \in \mathbb{C}(\Sigma)$, and scripted letters such as $\mathcal{A}, \mathcal{B}, \mathcal{X}, \mathcal{Y}$, etc.., are generally used. For example, we might write $\mathcal{X} = \mathbb{C}(\{0, 1\}^n)$ to indicate that the space \mathcal{X} is indexed by the set $\{0, 1\}^n$, and simply use the symbol \mathcal{X} to refer to that space from that point on. Although it will seldom be necessary, we may also write $\mathbb{C}(\Sigma)^\dagger$ or \mathcal{X}^\dagger (for example) to refer to the space of all row vectors (or bra vectors) $\langle \psi |$, for $|\psi\rangle \in \mathbb{C}(\Sigma)$ or $|\psi\rangle \in \mathcal{X}$. You will also see things written with a start * instead of a dagger \dagger sometimes, as in \mathcal{X}^* .

Also similar to before, when we consider a particular quantum system we assume that it has some finite set Σ of associated classical states. We will typically use the term register from now on to refer to abstract physical devices (such as qubits or

collection of qubits). Associated with any register having classical state Σ is the vector space $\mathbb{C}(\Sigma)$. It is sometimes helpful, but certainly not necessary, to use the same letter (in different fonts) to refer to a register and its associated space. for example, register X may have associated space \mathcal{X} .

33.1 Density Matrices

So far nothing is new except a little bit of notation. In order to describe what really is new, it is helpful to start with a special case. Suppose that in the “old” representation, a register X having classical state set Σ is in a quantum state $|\psi\rangle \in \mathcal{X}$ for $\mathcal{X} = \mathbb{C}(\Sigma)$. The “new” way of representing this state will be:

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

We have seen objects like this before (in the analysis of Grover’s algorithm). It is effectively a matrix. For example, suppose $\Sigma = \{0, 1\}$ and $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Then

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad \text{and} \quad \langle\psi| = (\bar{\alpha} \quad \bar{\beta})$$

so

$$|\psi\rangle \langle\psi| = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} (\bar{\alpha} \quad \bar{\beta}) = \begin{pmatrix} \alpha\bar{\alpha} & \alpha\bar{\beta} \\ \bar{\alpha}\beta & \bar{\beta}\bar{\beta} \end{pmatrix} = \begin{pmatrix} |\alpha|^2 & \alpha\bar{\beta} \\ \bar{\alpha}\beta & |\beta|^2 \end{pmatrix}$$

This is called a *density matrix* or *density operator*. (the term operation usually just means a linear map from a space to itself). Not all density operators have the special form $|\psi\rangle \langle\psi|$ for some unit vector $|\psi\rangle$, as we will see. All of the quantum states we have considered so far in the course have actually been of this special kind of state.

Suppose we have a probability distribution (p_1, \dots, p_k) , for some positive integer k , as well as unit vectors $|\psi_1\rangle, \dots, |\psi_k\rangle \in \mathcal{X}$, where $\mathcal{X} = \mathbb{C}(\Sigma)$ is the space corresponding to some register X. Somebody randomly chooses $j \in \{1, \dots, k\}$ according to the probability distribution (p_1, \dots, p_k) , prepares the register X in the state $|\psi_j\rangle$ for the chosen j , and then hands you X without telling you the value of j . The collection $\{(p_1, |\psi_1\rangle), \dots, (p_k, |\psi_k\rangle)\}$ that describes the different possible states $|\psi_j\rangle$ along with their associated probabilities is called a *mixture*. How do you represent this beyond specifying the mixture. in the “old” representation, there is no convenient way to do this beyond specifying the mixture. In the “new” representation, the density matrix corresponding to the above mixture is:

$$\rho = \sum_{j=1}^k p_j |\psi_j\rangle \langle\psi_j|$$

In other words it is meaningful to take a weighted average of the pure states $|\psi_j\rangle\langle\psi_j|$. More precisely, suppose a quantum system is in one of a number of states $|\psi_i\rangle$, where i is an index, with respective probabilities p_i . We shall call $\{p_i, |\psi_i\rangle\}$ an ensemble of pure states.

Example 33.1.1. Suppose Alice has a qubit A. She flips a fair coin: if the result is heads she prepares A in the state $|0\rangle$, and if the result is Tails she prepares A in the state $|1\rangle$. She gives Bob the qubit without revealing the result of the coin-flip. Bob's knowledge of the qubit is described by the density matrix

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

Example 33.1.2. Suppose Alice has a qubit A as in the previous example. As before she flips a fair coin, but now if the result is Heads she prepares A in the state $|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, and if the result is Tails she prepares A in the state $|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$. Bob's knowledge of the qubit is now described by the density matrix

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

Same as before!.

The previous two examples demonstrate that different mixtures can yield precisely the same density matrix. This is not an accident, but rather is one of the strengths of the density matrix formalism. A given density matrix in essence represents a perfect description of the state of a quantum system - two different mixtures can be distinguished (in a statistical sense) if and only if they yield different density matrices. Whether one uses the density operator language or the state vector language is a matter of taste, since both give the same results; however it is sometimes much easier to approach problems from one point of view rather than the other.

Typically, lower case Greek letters such as ρ, ξ, σ and τ are used to denote density matrices. The state of a system that corresponds to a given density matrix is called a mixed state.

A quantum system whose state $|\psi\rangle$ is known exactly is said to be in a *pure state*. In this case density operator is simply $\rho = |\psi\rangle\langle\psi|$. Otherwise, ρ is in a *mixed state*; it is said to be a mixture of the different pure states in the ensemble for ρ . Note: sometimes people may use the term ‘mixed’ state as a catch-all to include both pure and mixed quantum states. The term ‘pure state’ is often used in reference to a state vector $|\psi\rangle$, to distinguish it from a density operator ρ .

Finally, imagine a quantum system is prepared in the state ρ_i with probability p_i . It is not difficult to convince yourself that the system may be described by the density matrix $\sum_i p_i \rho_i$. A proof of this is to suppose that ρ_i arises from some ensemble $\{p_{ij}, |\psi_{ij}\rangle\}$ (note that i is fixed) of pure states, so the probability for being in the state $|\psi_{ij}\rangle$ is $p_i p_{ij}$. The density matrix for the system is thus

$$\begin{aligned}\rho &= \sum_{ij} p_i p_{ij} |\psi_{ij}\rangle \langle \psi_{ij}| \\ &= \sum_i p_i \rho_i\end{aligned}$$

where we have used the definition $\rho_i = \sum_j p_{ij} |\psi_{ij}\rangle \langle \psi_{ij}|$. We say that ρ is a mixture of the state ρ_i with probabilities p_i . This concept of a mixture comes up repeatedly in the analysis of problems like quantum noise, where the effect of the noise is to introduce ignorance into our knowledge of the quantum state. A simple example is provided by the measurement scenario. Imagine that, for some reason our record of the result m of the measurement was lost. We would have a quantum system in the state ρ_m with probability $p(m)$, but would no longer know the actual value of m . The state of such a quantum system would therefore be described by the density operator

$$\begin{aligned}\rho &= \sum_m p(m) \rho_m \\ &= \sum_m \text{Tr}(M_m^\dagger M_m \rho) \frac{M_m \rho M_m^\dagger}{\text{tr}(M_m^\dagger M_m \rho)} \\ &= \sum_m M_m \rho M_m^\dagger\end{aligned}$$

General properties of density matrices: We now move away from this description to develop an intrinsic characterization of density operator that does not rely on an ensemble interpretation. This allows us to complete the program of giving a description of quantum mechanics that does not take as its foundation the state vector.

1. Trace Condition: The trace of a density matrix is 1

(The trace of a matrix is sum of its diagonal entries)

In fact, the diagonal entries describe precisely the probability distribution that would result if the system in question were measured (with respect to the restricted type of measurement we have so far considered).

The fact that the trace is a linear function together with the useful formula $\text{Tr}(AB) = \text{Tr}(BA)$ (which holds for any choice of matrices A and B for which the product AB is square) makes the above fact easy to verify for any given mixture:

$$\text{Tr} \left(\sum_{j=1}^k p_j |\psi_j\rangle \langle \psi_j| \right) = \sum_{j=1}^k p_j \text{Tr}(|\psi_j\rangle \langle \psi_j|) = \sum_{j=1}^k p_j \text{Tr}(\langle \psi_j | |\psi_j\rangle) = \sum_{j=1}^k p_j \langle \psi_j | \psi_j \rangle = 1$$

2. Positivity Condition: Every density matrix is positive semi-definite.

In general, a square matrix A is positive semi-definite if $\langle \psi | A \psi \rangle$ is a non-negative real number for every vector $|\psi\rangle$. An equivalent definition is that A is positive semi-definite if (i) $A = A^\dagger$ (i.e., A is Hermitian), and (ii) all eigenvalues of A are non-negative real numbers.

Again this condition is easy to check for mixtures: if

$$\rho = \sum_{j=1}^k p_j |\psi_j\rangle \langle \psi_j|$$

and $|\phi\rangle$ is any vector, we have

$$\langle \phi | \rho | \phi \rangle = \sum_{j=1}^k p_j \langle \phi | \psi_j \rangle \langle \psi_j | \phi \rangle = \sum_{j=1}^k p_j |\langle \phi | \psi_j \rangle|^2 \geq 0$$

You can interpret the above facts as the defining properties of density matrices: by definition, a density matrix is any matrix that is positive semi-definite and has trace equal to 1. Given such a matrix, it is possible to find a mixture that yields the given density matrix by letting p_1, \dots, p_k be the nonzero eigenvalues and $|\psi_1\rangle, \dots, |\psi_k\rangle$. Suppose ρ is any operator satisfying the trace and positivity conditions. Since ρ is positive, it must have a spectral decomposition

$$\rho = \sum_j \lambda_j |j\rangle \langle j|$$

where the vectors $|j\rangle$ are orthogonal, and λ_j are real, non-negative eigenvalues of ρ . From the trace condition we see that $\sum_j \lambda_j = 1$. Therefore, a system in the state $|j\rangle$ with probability λ_j will have density operator ρ . That is, the ensemble $\{\lambda_j, |j\rangle\}$ is an ensemble of states giving rise to the density operator ρ . This theorem provides a characterization of density operators that is intrinsic to the operator itself: we can define a density operator to be a positive operator ρ which has trace equal to one. Making this definition allows us to reformulate the postulates of quantum mechanics in the density operator picture.

33.2 Operations on Density matrices

In our “old” description of quantum information, a unitary operation U applied to a state $|\psi\rangle$ resulted in the state $U|\psi\rangle$. In the density matrix description, a unitary operation U applied to a pure state $|\psi\rangle\langle\psi|$ results in the density matrix

$$U|\psi\rangle\langle\psi|U^\dagger$$

This is consistent with the observation that $(U|\psi\rangle)^\dagger = \langle\psi|U^\dagger$. More generally, applying U to the mixed state ρ results in the state $U\rho U^\dagger$. you can easily that that the two required conditions of density matrices are necessarily met by this new matrix.

We can, however, consider a much more general set of possible operations than just unitary operations. Any operation Φ that can be written as

$$\Phi(\rho) = \sum_{j=1}^k A_j \rho A_j^\dagger$$

for some collection of matrices A_1, \dots, A_k satisfying

$$\sum_{j=1}^k A_j^\dagger A_j = I$$

represents an operation that can (in an idealized sense) be physically implemented. Such operations are called admissible operations. (There are several other names that are used as well, such as completely positive trace preserving operations and other variations on these words.)

If ρ is a matrix and Φ is admissible, then $\Phi(\rho)$ is also a density matrix. In fact a somewhat stronger property holds, which is that if Φ is applied to just part of a larger system whose state is described by some density matrix, then the resulting state will also be described by a density matrix.

Example 33.2.1. Suppose we have a single qubit X. Consider the operations that corresponds to measuring the qubit and forgetting the result. An admissible operation that describes this process is given by

$$A_0 = |0\rangle\langle 0|$$

$$A_1 |1\rangle\langle 1|$$

$$\Phi(\rho) = \sum_{j=0}^1 A_j \rho A_j^\dagger = |0\rangle \langle 0| \rho |0\rangle \langle 0| + |1\rangle \langle 1| \rho |1\rangle \langle 1|$$

First let us check that this is a valid admissible operation:

$$\sum_{j=0}^1 A_j A_j^\dagger = |0\rangle \langle 0| |0\rangle \langle 0| + |1\rangle \langle 1| |1\rangle \langle 1| = |0\rangle \langle 0| + |1\rangle \langle 1| = I$$

It satisfies the required property, so indeed it is admissible.

What does it do to the state $\alpha |0\rangle + \beta |1\rangle$, for example? First we need to represent $|\psi\rangle$ as a density matrix: $\rho = (|\psi\rangle \langle \psi|)$. Now

$$\begin{aligned}\Phi(\rho) &= \Phi(|\psi\rangle \langle \psi|) \\ &= |0\rangle \langle 0| \psi \rangle \langle \psi| |0\rangle \langle 0| + |1\rangle \langle 1| \psi \rangle \langle \psi| |1\rangle \langle 1| \\ &= |\langle 0|\psi\rangle|^2 |0\rangle \langle 0| + |\langle 1|\psi\rangle|^2 |1\rangle \langle 1| \\ &= |\alpha|^2 |0\rangle \langle 0| + |\beta|^2 |1\rangle \langle 1|\end{aligned}$$

In terms of matrices:

$$\begin{pmatrix} |\alpha|^2 & \alpha\bar{\beta} \\ \bar{\alpha}\beta & |\beta|^2 \end{pmatrix} \xrightarrow{\Phi} \begin{pmatrix} |\alpha|^2 & 0 \\ 0 & |\beta|^2 \end{pmatrix}$$

In general, off diagonal entries get zeroed out.

In general, admissible operation do not need to preserve the sizes of quantum systems. For example, one might consider the situation in which one of a collection of qubits is lost or somehow destroyed. More notation will help to discuss this issue in greater specificity.

Given two spaces \mathcal{X} and \mathcal{Y} , we let

$$L(\mathcal{X}, \mathcal{Y})$$

denote the set of all linear mappings from \mathcal{X} to \mathcal{Y} . The shorthand $L(\mathcal{X})$ is used to mean $L(\mathcal{X}, \mathcal{X})$. Also let

$$D(\mathcal{X})$$

denote the set of all density matrices on \mathcal{X} (so that $D(\mathcal{X}) \subset L(\mathcal{X})$). For example, if X is a quantum register with classical set Σ and $\mathcal{X} = \mathbb{C}(\Sigma)$, then any mixed state of the register X is represented by some element of $D(\mathcal{X})$.

Now, if we have a collection of mappings (or matrices) $A_1, \dots, A_k \in L(\mathcal{X}, \mathcal{Y})$ that satisfy

$$\sum_{j=1}^k A_j^\dagger A_j = I$$

(the density matrix in $L(\mathcal{X})$) then the admissible operation Φ is defined by

$$\Phi(\rho) = \sum_{j=1}^k A_j \rho A_j^\dagger$$

maps elements of $D(\mathcal{X})$ to elements of $D(\mathcal{Y})$.

Example 33.2.2. Let us suppose that we have two qubits: X and Y . We will consider the admissible operation that corresponds to discarding the second qubit. Thus, once the operation has been performed, we will be left with a single qubit X . The vector space corresponding to X will be \mathcal{X} and the space corresponding to Y will be \mathcal{Y} .

Now if Φ is to describe the operation of discarding the second qubit, then we must have that $\Phi(\rho) \in D(\mathcal{X})$ whenever $\rho \in D(\mathcal{X} \otimes \mathcal{Y})$. This means that if

$$\Phi(\rho) = \sum_{j=1}^k A_j \rho A_j^\dagger$$

for some choice of matrices A_1, \dots, A_k then these matrices must come from the set $L(\mathcal{X} \otimes \mathcal{Y}, \mathcal{X})$. this means that they must be 2×4 matrices. Let

$$A_0 = I \otimes \langle 0 |, \quad A_1 = I \otimes \langle 1 |$$

where I denotes the identity operator on the first qubit, and define Φ by

$$\Phi(\rho) = \sum_{j=0}^1 A_j \rho A_j^\dagger = A_0 \rho A_0^\dagger + A_1 \rho A_1^\dagger$$

for all ρ . writing A_0 and A_1 in ordinary matrix notation gives

$$A_0 = I \otimes \langle 0 | = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes (1 \ 0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$A_1 = I \otimes \langle 1 | = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes (0 \ 1) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

To see that Φ is admissible, we compute:

$$\begin{aligned} A_0^\dagger A_0 + A_1^\dagger A_1 &= (I \otimes |0\rangle)(I \otimes \langle 0|) + (I \otimes |1\rangle)(I \otimes \langle 1|) \\ &= I \otimes |0\rangle \langle 0| + I \otimes |1\rangle \langle 1| \\ &= I \otimes (|0\rangle \langle 0| + |1\rangle \langle 1|) \\ &= I \otimes I \\ &= I_{\mathcal{X} \otimes \mathcal{Y}} \end{aligned}$$

(The subscript on the identity operator in the last line is just representing what space the identity is action on.)

Let us now consider the effect of this operation on couple of states. First suppose that (X, Y) is in the state $\xi \otimes \sigma$ for two 2×2 density matrices ξ and σ . Just as for vectors in the “old” description of quantum information, we view states of the form $\xi \otimes \sigma$ as representing completely uncorrelated qubits. What do you expect the state to be if you discard the second qubit? Now let us check:

$$\begin{aligned}\Phi(\xi \otimes \sigma) &= A_0(\xi \otimes \sigma)A_0^\dagger + A_1(\xi \otimes \sigma)A_1^\dagger \\ &= (I \otimes \langle 0|)(\xi \otimes \sigma)(I \otimes |0\rangle) + (I \otimes \langle 1|)(\xi \otimes \sigma)(I \otimes |1\rangle) \\ &= \xi \otimes \langle 0|\sigma|0\rangle + \xi \otimes \langle 1|\sigma|1\rangle \\ &= (\langle 0|\sigma|0\rangle + \langle 1|\sigma|1\rangle)\xi \\ &= Tr(\sigma)\xi \\ &= \xi\end{aligned}$$

Indeed this what you presumably expected.

Now we shall see that happens when this operation is applied to entangled qubits.

The previous example can be generalized to describe the operation that corresponds to discarding part of a system. Suppose X and Y are registers with corresponding spaces \mathcal{X} and \mathcal{Y} . A mixed state of these two registers is represented by some element of $D(\mathcal{X} \otimes \mathcal{Y})$. If, say the register Y is discarded, we will be left with some mixed state of X that is represented by some element of $D(\mathcal{X})$. Specifically, if $\rho \in D(\mathcal{X} \otimes \mathcal{Y})$ is a density matrix representing the state of (X, Y) and Y is discarded, the resulting state of X is denoted by $Tr_{\mathcal{Y}}(\rho)$ the partial trace. (It is called the partial trace because when extended by linearity to arbitrary matrices it satisfies $Tr_{\mathcal{Y}}(X \otimes Y) = Tr(Y)X$ for all $X \in L(\mathcal{X})$ and $Y \in L(\mathcal{Y})$.) We also refer to the action corresponding to this operation as tracing out the space \mathcal{Y} . To express $Tr_{\mathcal{Y}}$ in the form of an admissible operation, let Σ denote the set of classical states of Y . Then

$$Tr_{\mathcal{Y}}(\rho) = \sum_{a \in \Sigma} (I \otimes \langle a|)\rho(I \otimes |a\rangle)$$

The partial trace is a particularly important admissible operation that we will continue discussing.

33.3 Postulates

It turns out that all the postulates of quantum mechanics can be reformulated in terms of the density operator language. The purpose of this section and the next is to explain how to perform this reformulation, and explain when it is useful.

33.3.1 Postulate 1: The State Postulate

Associated to any isolated physical system is a complex vector space with inner product (That is, a Hilbert space) known as the state space of the system. The system is completely described by its density operator, which is a positive operator ρ with trace one, acting on the state space of the system. If a quantum system is in the state ρ_i with probability p_i , then the density operator for the system is $\sum_i p_i \rho_i$.

33.3.2 Postulate 2: Evolution Postulate

The evolution of a closed system is described by a Unitary transformation. That is, the state ρ of the system at time t_1 is related to the state ρ' of the system at time t_2 by a unitary operator U which depends only on the times t_1 and t_2 .

Suppose evolution of a closed quantum system is described by the unitary operator U . If the system was initially in the state $|\psi_i\rangle$ with probability p_i then after the evolution has occurred the system will be in the state $U|\psi_i\rangle$ with probability p_i . Thus, the evolution of the density operator is described by the equation

$$\rho = \sum_i p_i |\psi_i\rangle \langle \psi_i| \xrightarrow{U} \sum_i p_i U |\psi_i\rangle \langle \psi_i| U^\dagger = U \rho U^\dagger$$

33.3.3 Postulate 3: Measurement Postulate

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

$$p(m) = \langle \psi | M_m^\dagger M_m | \psi \rangle = \text{Tr}(M_m^\dagger M_m \rho)$$

and the state of the system after the measurement is

$$\frac{M_m |\psi\rangle \langle \psi| M_m^\dagger}{\sqrt{p(m)} \sqrt{p(m)}} = \frac{M_m \rho M_m^\dagger}{\text{Tr}(M_m^\dagger M_m \rho)}$$

The measurement operators satisfy the completeness equation,

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

Measurements are also easily described in the density operator language. Suppose we perform a measurement described by measurement operators M_m . If the initial state was $|\psi_i\rangle$, then the probability of getting result m is

$$p(m|i) = \langle\psi_i|M_m^\dagger M_m|\psi_i\rangle = Tr(M_m^\dagger M_m |\psi_i\rangle \langle\psi_i|)$$

Using the cyclic property of trace of a matrix. By the law of total probability (see Appendix) the probability of obtaining the result m is

$$\begin{aligned} p(m) &= \sum_i p(m|i)p_i \\ &= \sum_i p_i Tr(M_m^\dagger M_m |\psi_i\rangle \langle\psi_i|) \\ &= Tr(M_m^\dagger M_m \sum_i p_i |\psi_i\rangle \langle\psi_i|) \\ &= Tr(M_m^\dagger M_m \rho) \end{aligned}$$

The density operator of the system after the measurement result m is

$$|\psi_i^m\rangle = \frac{M_m |\psi_i\rangle}{\sqrt{\langle\psi_i|M_m^\dagger M_m|\psi_i\rangle}}$$

Thus, after a measurement which yields the result m we have an ensemble of states $|\psi_i^m\rangle$ with respective probabilities $p(i|m)$. The corresponding density operator ρ_m is therefore

$$\rho_m = \sum_i p(i|m) |\psi_i^m\rangle \langle\psi_i^m| = \sum_i p(i|m) \frac{M_m |\psi_i\rangle \langle\psi_i| M_m^\dagger}{Tr(M_m^\dagger M_m |\psi_i\rangle)}$$

But by elementary probability theory, $p(i|m) = p(m,i)/p(m) = p(m|i)p_i/p(m)$. Substituting, we obtain

$$\begin{aligned} \rho_m &= \sum_i p_i \frac{M_m |\psi_i\rangle \langle\psi_i| M_m^\dagger}{Tr(M_m^\dagger M_m \rho)} \\ &= \frac{M_m \rho M_m^\dagger}{Tr(M_m^\dagger \rho M_m)} \end{aligned}$$

Thus, the basic postulates of quantum mechanics related to unitary evolution and measurements can be rephrased in the language of density operators.

33.3.4 Postulate 4: Composite Systems

The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. moreover, if we have systems numbered 1 through n , and system number i is prepared in the state ρ_i , then the joint state of the total system is $\rho_1 \otimes \rho_2 \otimes \dots \otimes \rho_n$.

These reformulations of the fundamental postulates of quantum mechanics in terms of density operator are mathematically equivalent to the description in terms of the state vector. As a way of thinking about quantum mechanics, the density operator approach really shines for two applications: the description of quantum systems whose state is not known, and the description of subsystems of a composite quantum system.

Example 33.3.1. Let ρ be a density operator. Show that $Tr(\rho^2) \leq 1$, with equality if and only if ρ is a pure state.

Recall that $\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|$. Now,

$$\begin{aligned} Tr(\rho^2) &= Tr\left(\left(\sum_i p_i |\psi_i\rangle\langle\psi_i|\right)\left(\sum_j p_j |\psi_j\rangle\langle\psi_j|\right)\right) \\ &= Tr\left(\sum_i \sum_j p_i p_j |\psi_i\rangle\langle\psi_i|\psi_j\rangle\langle\psi_j|\right) \\ &= \sum_i \sum_j p_i p_j Tr(|\psi_i\rangle\langle\psi_i|\psi_j\rangle\langle\psi_j|) \\ &= \sum_i \sum_j p_i p_j Tr(\langle\psi_j|\psi_i\rangle\langle\psi_i|\psi_j\rangle) \\ Tr(\rho^2) &= \sum_i \sum_j p_i p_j |\langle\psi_i|\psi_j\rangle|^2 \end{aligned}$$

Now using the Cauchy-Schwarz Inequality, we know, $|\langle\psi_i|\psi_j\rangle|^2 \leq \langle\psi_i|\psi_i\rangle\langle\psi_j|\psi_j\rangle$, we

get,

$$\begin{aligned}
\sum_{i,j} p_i p_j |\langle \psi_i | \psi_j \rangle|^2 &\leq \sum_{i,j} p_i p_j \langle \psi_i | \psi_i \rangle \langle \psi_j | \psi_j \rangle \\
&\leq \sum_{i,j} p_i p_j \\
&\leq \left(\sum_i p_i \right) \left(\sum_j p_j \right) \\
&\leq 1 \\
Tr(\rho^2) &\leq 1
\end{aligned}$$

Clearly, the equality holds here if and only if $|\psi\rangle = |\psi_j\rangle \quad \forall i, j$ upto a global phase factor i.e. there is only one $|\psi\rangle$ that occurs with probability 1.

It is tempting (and surprisingly common) fallacy to suppose that the eigenvalues and eigenvectors of a density matrix have some special significance with regard to the ensemble of quantum states represented by that density matrix. For example, one might suppose that a quantum system with density matrix

$$\rho = \frac{3}{4} |0\rangle \langle 0| + \frac{1}{4} |1\rangle \langle 1|$$

must be in the state $|0\rangle$ with probability $3/4$ and in the state $|1\rangle$ with probability $1/4$. However, this is not necessarily the case. Suppose we define

$$|a\rangle = \sqrt{\frac{3}{4}} |0\rangle + \sqrt{\frac{1}{4}} |1\rangle$$

$$|b\rangle = \sqrt{\frac{3}{4}} |0\rangle - \sqrt{\frac{1}{4}} |1\rangle$$

and the quantum system is prepared in the state $|a\rangle$ with probability $1/2$ and in the state $|b\rangle$ with probability $1/2$. Then it is easily checked that the corresponding density matrix is

$$\rho = \frac{1}{2} |a\rangle \langle a| + \frac{1}{2} |b\rangle \langle b| = \frac{3}{4} |0\rangle \langle 0| + \frac{1}{4} |1\rangle \langle 1|$$

That is, two different ensembles of quantum states give rise to the same density matrix. In general, the eigenvectors and eigenvalues of a density matrix just indicate

one of many possible ensembles that may give rise to a specific density matrix, and there is no reason to suppose it is an especially privileged ensemble.

That is, two different ensembles of quantum states give rise to the same density matrix. In general, eigenvalues and eigenvectors of a density matrix indicate one of many possible ensembles that may give rise to a specific density matrix, and there is no reason to suppose it is an especially privileged ensemble.

A natural question to ask in the light of this discussion is what class of ensemble does give rise to a particular density matrix? The solution to this problem, which we now give, has surprisingly many applications in quantum computation and quantum information, notably in the understanding of quantum noise and quantum error-correction. For the solution it is convenient to make use of vectors $|\tilde{\psi}_i\rangle$ which may not be normalized to unit length. We say the set $|\tilde{\psi}_i\rangle$ generates the operator $\rho = \sum_i |\tilde{\psi}_i\rangle\langle\tilde{\psi}_i|$, and thus the connection to the unusual ensemble picture of density operators is expressed by the equation $|\tilde{\psi}_i\rangle = \sqrt{p_i}|\psi_i\rangle$. When do two sets of vectors, $|\tilde{\psi}_i\rangle$ and $|\tilde{\varphi}_j\rangle$ generate the same operator ρ ? The solution to this problem will enable us to answer the question of what ensemble gives rise to a given density matrix.

Theorem 33.3.1. (*Unitary freedom in the ensemble for density matrices*)
The sets $|\tilde{\psi}_i\rangle$ and $|\tilde{\varphi}_j\rangle$ generate the same density matrix if and only if

$$|\tilde{\psi}_i\rangle = \sum_j u_{ij} |\tilde{\varphi}_j\rangle$$

where u_{ij} is a unitary matrix of complex numbers, with indices i and j , and we ‘pad’ whichever set of vectors $|\tilde{\psi}_i\rangle$ or $|\tilde{\varphi}_j\rangle$ is smaller with additional vectors 0 so that the two sets have the same number of elements. As a consequence of the theorem, note that $\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| = \sum_j q_j |\varphi_j\rangle\langle\varphi_j|$ for normalized states $|\psi_i\rangle, |\varphi_j\rangle$ and probability distributions p_i and q_j if and only if

$$\sqrt{p_i}|\psi_i\rangle = \sum_j u_{ij}\sqrt{q_j}|\varphi_j\rangle$$

for some unitary matrix u_{ij} , and we may pad the smaller ensemble with entries having probability zero in order to make the two ensembles the same size. Thus, the theorem characterizes the freedom in the ensembles $\{p_i, |\psi_i\rangle\}$ giving rise to a given density matrix ρ . indeed, it is easily checked that our earlier example of a density matrix with two different decomposition, arises as a special case of this general result.

Proof. Suppose $|\tilde{\psi}_i\rangle = \sum_j u_{ij} |\tilde{\varphi}_j\rangle$ for some unitary u_{ij} . Then

$$\begin{aligned}\sum_i |\tilde{\psi}_i\rangle &= \sum_{ijk} u_{ij} u_{ik}^* |\tilde{\varphi}_j\rangle \langle \tilde{\varphi}_k| \\ &= \sum_{jk} \left(\sum_i u_{ki}^\dagger u_{ij} \right) |\tilde{\varphi}_j\rangle \langle \tilde{\varphi}_k| \\ &= \sum_{jk} \delta_{kj} |\tilde{\varphi}_j\rangle \langle \tilde{\varphi}_k| \\ &= \sum_j |\tilde{\varphi}_j\rangle \langle \tilde{\varphi}_j|\end{aligned}$$

which shows that $|\tilde{\psi}_i\rangle$ and $|\tilde{\varphi}_j\rangle$ generate the same operator.

Conversely, suppose

$$A = \sum_i |\tilde{\psi}_i\rangle \langle \tilde{\psi}_i| = \sum_j |\tilde{\varphi}_j\rangle \langle \tilde{\varphi}_j|$$

Let $A = \sum_k \lambda_k |k\rangle \langle k|$ be a decomposition for A such that the states $|k\rangle$ are orthonormal, and the λ_k are strictly positive. Our strategy is to relate the state $|\tilde{\psi}_i\rangle$ to the states $|\tilde{k}\rangle = \sqrt{\lambda_k} |k\rangle$, and similarly relate the states $|\tilde{\varphi}_j\rangle$ to the states $|\tilde{k}\rangle$. Combining the two relations will give the result. Let $|\psi\rangle$ be any vector orthonormal to the space spanned by $|\tilde{k}\rangle$, so $\langle\psi|\tilde{k}\rangle \langle\tilde{k}|\psi\rangle = 0$ for all k , and thus we see that

$$0 = \langle\psi|A|\psi\rangle = \sum_i \langle\psi|\tilde{\psi}_i\rangle \langle\tilde{\psi}_i|\psi\rangle = \sum_i |\langle\psi|\tilde{\psi}_i\rangle|^2$$

Thus $\langle\psi|\tilde{\psi}_i\rangle = 0$ for all i and all $|\psi\rangle$ orthonormal to the space spanned by the $|\tilde{k}\rangle$. It follows that each $|\tilde{\psi}_i\rangle$ can be expressed as a linear combination of the $|\tilde{k}\rangle$, $|\tilde{\psi}_i\rangle = \sum_k c_{ik} |\tilde{k}\rangle$. Since $A = \sum_k |\tilde{k}\rangle \langle \tilde{k}| = \sum_i |\tilde{\psi}_i\rangle \langle \tilde{\psi}_i|$ we see that

$$\sum_k |\tilde{k}\rangle \langle \tilde{k}| = \sum_{kl} \left(\sum_i c_{ik} c_{il}^* \right) |\tilde{k}\rangle \langle \tilde{l}|$$

The operators $|\tilde{k}\rangle \langle \tilde{l}|$ are easily seen to be linearly independent, and thus it must be that $\sum_i c_{ik} c_{il}^* = \delta_{kl}$. This ensures that we may append extra columns to c to obtain a unitary matrix v such that $|\tilde{\psi}_i\rangle = \sum_k v_{ik} |\tilde{k}\rangle$, where we have appended zero vectors to the list of $|\tilde{k}\rangle$. Similarly, we can find a unitary matrix w such that $|\tilde{\varphi}_j\rangle = \sum_k w_{jk} |\tilde{k}\rangle$. Thus $|\tilde{\psi}_i\rangle = \sum_j u_{ij} |\tilde{\varphi}_j\rangle$, where $u = vw^\dagger$ is unitary. \square

Example 33.3.2. (Bloch sphere for mixed states) The Bloch sphere picture for pure states of a single qubit

33.4 The Reduced Density Operator

Perhaps the deepest application of the density operator is as a descriptive tool for sub-systems of a composite quantum systems. Such a description is provided by the reduced density operator, which is the subject of this section. The reduced density operator is so useful as to be virtually indispensable in the analysis of composite quantum systems.

Suppose we have physical systems A and B , whose states are described by a density operator ρ^{AB} . The reduced density operator for system A is defined by

$$\rho^A \equiv \text{tr}_B(\rho^{AB})$$

where tr_B is a map of operators known as the partial trace over system B . The partial trace is defined by

$$\text{tr}_B(|a_1\rangle\langle a_2| \otimes |b_1\rangle\langle b_2|) \equiv |a_1\rangle\langle a_1| \text{tr}(|b_1\rangle\langle b_1|)$$

where $|a_1\rangle$ and $|a_2\rangle$ are any two vectors in the state space of A , and $|b_1\rangle$ and $|b_2\rangle$ are any two vectors in the state space of B . The trace operation appearing on the right hand side is the usual trace operation for system B , so $\text{tr}(|b_1\rangle\langle b_2|) = \langle b_1|b_2\rangle$. We have defined the partial trace operation only on a special subclass of operators AB ; the specification is completed by requiring in addition to above equation that the partial trace be linear in its input.

It is not obvious that the reduced density operator for system A is in any sense a description for the state of system A . The physical justification for making this identification is that the reduced density operator provides the correct measurement statistics for measurements made on system A . This is e

Example 33.4.1. (Block sphere for mixed states) The Bloch sphere description has an important generalization for mixed states as follows:

1. Show that an arbitrary density matrix for a mixed state qubit may be written as

$$\rho = \frac{I + \mathbf{r} \cdot \boldsymbol{\sigma}}{2}$$

where \mathbf{r} is a real three-dimensional vector such that $\|\mathbf{r}\| \leq 1$. This vector is known as the Block vector for the state ρ .

2. What is the Bloch vector representation for the state $\rho = I/2$.
3. Show that a state ρ is pure if and only if $\|\mathbf{r}\| = 1$.

4. Show that for pure state the description of the Bloch vector we have given coincides with the previous definition.

Since $\{I, X, Y, Z\}$ form an basis the vector space of single-qubit linear operators, we can write (for any ρ , regardless of whether it is a density operator or not):

$$\rho = a_1 I + a_2 X + a_3 Y + a_4 Z$$

for constants $a_1, a_2, a_3, a_4 \in \mathbb{C}$. Since ρ is a Hermitian operator, we find that each of these constants are actually real, as

$$a_1 I + a_2 X + a_3 Y + a_4 Z = \rho = \rho^\dagger = a_1^* I^\dagger + a_2^* X^\dagger + a_3^* Y^\dagger + a_4^* Z^\dagger = a_1^* I + a_2^* X + a_3^* Y + a_4^* Z$$

Now, we require that $\text{tr}(\rho) = 1$ for any density operator, hence:

$$\text{tr}(\rho) = \text{tr}(a_1 I + a_2 X + a_3 Y + a_4 Z) = a_1 \text{tr}(I) + a_2 \text{tr}(X) + a_3 \text{tr}(Y) + a_4 \text{tr}(Z) = 2a_1 = 1$$

from which we obtain that $a_1 = \frac{1}{2}$. Note that in the second equality we use the linearity of the trace, and in the third equality we use that $\text{Tr}(I) = 2$ and $\text{Tr}(\sigma_i) = 0$ for $i \in \{1, 2, 3\}$. Calculating ρ^2 , we have that:

$$\begin{aligned} \rho^2 &= \frac{1}{4} I + \frac{a_2}{2} X + \frac{a_3}{2} Y + \frac{a_4}{2} Z + \frac{a_2}{2} X + a_2^2 X^2 + a_2 a_3 X Y + a_2 a_4 X Z \\ &\quad + \frac{a_3}{2} Y + a_3 a_2 X U X + a_3^2 Y^2 + a_3 a_4 Y Z + \frac{a_4}{2} Z + a_4 a_2 Z X + a_4 a_3 Z Y + a_4^2 Z^2 \end{aligned}$$

Now, using that $\{\sigma_i, \sigma_j\} = 0$ for $i, j \in \{1, 2, 3\}, i \neq j$ and that $\sigma_i^2 = I$ for any $i \in \{1, 2, 3\}$, the above simplifies to:

$$\rho^2 = \left(\frac{1}{4} + a_2^2 + a_3^2 + a_4^2 \right) I + a_2 X + a_3 Y + a_4 Z$$

Taking the trace of ρ^2 we have that:

$$\text{tr}(\rho^2) = \left(\frac{1}{4} + a_2^2 + a_3^2 + a_4^2 \right) I + a_2 X + a_3 Y + a_4 Z$$

From the previous exercise we know that $\text{tr}(\rho^2) \leq 1$, so:

$$2 \left(\frac{1}{4} + a_2^2 + a_3^2 + a_4^2 \right) \leq 1 \implies a_2^2 + a_3^2 + a_4^2 \leq \frac{1}{4} \implies \sqrt{a_2^2 + a_3^2 + a_4^2} \leq \frac{1}{2}$$

Hence we can write:

$$\rho = \frac{I + r_x X + r_y Y + r_z Z}{2} = \frac{I + r \cdot \sigma}{2}$$

with $\|r\| \leq 1$. The Bloch sphere representation for the state $\rho = \frac{I}{2}$ is the above form with $r = 0$. This vector corresponds to the center of the Bloch sphere, which is a maximally mixed state $\text{tr}(\rho^2)$ is minimized, with $\text{tr}(\rho^2) = \frac{1}{2}$.

For the previous calculations we know that for any ρ

$$\text{tr}(\rho^2) = 2 \left(\frac{1 + r_z^2 + r_y^2 + r_z^2}{4} \right) = \frac{1 + r_x^2 + r_y^2 + r_z^2}{2}$$

if $\|r\| = 1$, then $r_x^2 + r_y^2 + r_z^2 = 1$. Hence, $\text{tr}(\rho^2) = 1$ and ρ is pure. Conversely, suppose ρ is pure. Then, $\text{tr}(\rho^2) = 1$, so:

$$\frac{1 + r_x^2 + r_y^2 + r_z^2}{2} = 1 \implies r_x^2 + r_y^2 + r_z^2 = 1 \implies \|r\| = 1$$

For the states that lie on the surface of the Bloch sphere, which we parameterized as

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

Calculating the density operator corresponding to $|\psi\rangle$, we have:

$$\rho = |\psi\rangle\langle\psi| =$$

Part V

Quantum Error Correction

Chapter 34

Classical Error Correction

Before going to Quantum Error correction we will just take a step back and look at the Classical Error correction in order to understand in general the ideas of error correction.

34.1 Background and Motivation

We now talk about the communication system Model. In Shanon's 1948 paper on "A mathematical Theory of Communication" he introduced a schematic diagram for a general communication system as shown in figure 34.1. By communication system we will mean a system of the type indicated schematically in figure 34.1. It consists of essentially five parts:

1. An information source which produces a message or a sequence of messages to be communicated to the receiving terminal. The message may be of various types: (a) A sequence of letters as in a telegraph or teletype system; (b) A single function of time $f(t)$ as in radio or telephony; (c) A function of time and other variables as in black and white television - here the message may be thought of as a function $f(x, y, t)$ of two space coordinates and time, the light intensity at point (x, y) at time t on a pickup tube plate; (d) Two or more functions of time, say $f(t), g(t), h(t)$ - this is the case in "three-dimensional" sound transmission or if the system is intended to service several individual channels in multiplex; (e) Several functions of several variables - in color television the message consists of three functions $f(x, y, t), g(x, y, t), h(x, y, t)$ defined in a three dimensional continuum - we may also think of these three functions as components of a vector field defined in the region- similarly, several black and

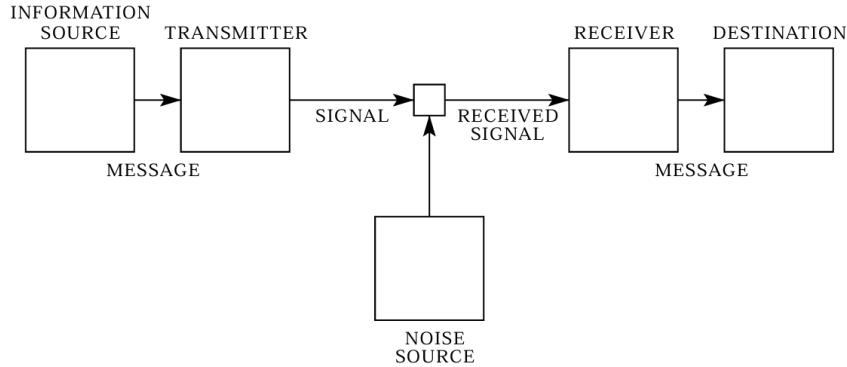


Figure 34.1: Schematic diagram for a general communication system

white television sources would produce “messages” consisting of a number of continuum of three variables; (f) Various combinations also occur, for example in television with an associated audio channel.

2. A transmitter which operates on the message in some way to produce a signal suitable for transmission over the channel. In telephony this operation consists merely of changing sound pressure into a proportional electrical current. In telegraphy we have an encoding operation which produces a sequence of dots, dashes and spaces on the channel corresponding to the message. In a multiplex PCM system the different speech functions must be sampled, compressed, quantized and encoded, and finally interleaved properly to construct the signal. Vocoder systems, television and frequency modulation are other examples of complex operations applied to the message to obtain the signal.
3. The channel is merely the medium used to transmit the signal from transmitter to receiver. It may be a pair of wires, a coaxial cable, a band of radio frequencies, a beam of light, etc.
4. The receiver ordinarily performs the inverse operation of that done by the transmitted, reconstruction the message from the signal.
5. The destination is the person (or thing) or whom the message is intended.

For analyzing, it is first necessary to represent the various elements involved as mathematical entities, suitably idealized from their physical counterparts. We may roughly classify communication systems into three main categories: discrete, continuous and mixed. By a discrete system we will mean one in which both the message

and the signal are a sequence of discrete symbols. A typical case is telegraphy where the message is a sequence of letters and the signal a sequence of dots, dashes and spaces. A continuous system is one in which the message and signal are both treated as continuous functions, e.g., radio or television. A mixed system is one in which both discrete and continuous variables appear, e.g., PCM transmission of speech.

We consider only the discrete case. This case has applications not only in communication theory, but also in the theory of computing machines, the design of telephone exchanges and other fields.

This naturally leads to the field of Coding theory. Coding is the representation using symbols, often 0s and 1s. What are the requirements for information transmission:

1. Cost of transmission should be low
2. Information should be transmitted reliably in the presence of channel noise
3. Information should be transmitted securely in the presence of an adversary/eavesdropper

The answer to each of these requirements is the following flavours of coding theory:

1. Source Coding: Enables to compress message to save on transmission.
2. Channel Coding: enables to send message reliably by introducing some mechanism to counter channel noise.
3. Secrecy Coding (aka Cryptography): Enables to encrypt message to transmit message securely in presence of an adversary.

Out of these three we will focus only on the first two parts i.e. Source Coding and Channel coding, which lead to Classical Error Correction. These two are closely related to Information theory. We assume that the Channel Noise follows a model known to both Information Source and Destination. This noise may be deterministic or random; if random, it follows a probabilistic model. The effect of noise is to introduce errors in the transmitted message and thus, the goal of channel coding is to reduce or eliminate the effects of channel noise.

Say Alice wishes to communicate with Bob a message made up of discrete symbols (0s and 1s). We will then use Source and Channel Coding to transmit the message. For the purpose of these notes, we ignore Secrecy coding aka Cryptography as Quantum Cryptography is in itself an entire field. We ignore the requirement

of information to be transmitted securely in the presence of an adversary/eavesdropper. We wish to transmit the information at low cost and reliably in the presence of channel noise which can be achieved through source coding and Channel coding. There are three types of channel codes:

1. **Error-Detecting Codes:** Allows Bob to detect errors in received message which is useful in random noise situations.
2. **Error-Correcting Codes:** Allows Bob to correct errors in received message which is useful in random noise situations.
3. **Constrained Codes:** Alice encodes the message in such a way as to prevent errors from corrupting the message which is useful in deterministic noise situations.

Important Note

All coding schemes work by adding redundancy to the message to compensate for errors i.e more symbols are transmitted than are in the original message.

34.2 Error-Detecting Code

Consider the following Error Detecting Code We wish to encode a message made up

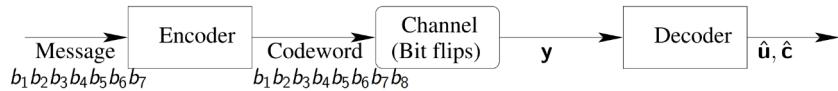


Figure 34.2: A simple Error-Detecting Code

of a 7-bit binary sequence $(b_1, b_2, b_3, b_4, b_5, b_6, b_7)$. As shown in figure 34.2 the encode adds 8th bit, called a parity bit, b_8 , so that, (b_1, b_2, \dots, b_8) contains an even number of 1s. Thus, the encode encodes the 7 bit message to an 8 bit message by adding a parity bit which is 1 if there are an odd number of 1 in the 7-bit message and 0 if there are an even bit of 1s in the message. Mathematically, we can write the following

$$\sum_{i=1}^8 b_i = 0 \pmod{2}$$

Note that the addition is done in modulo 2. The Channel noise model here is assume to randomly flip bits from $0 \rightarrow 1$ and $1 \rightarrow 0$. We now define Rate.

Definition 34.2.1. Rate: The number of message bits per coded bits.

For example, here the rate is $7/8$ because we have 7 message bits and the codeword after encoding is of 8 bits. Now consider the following example for a simple error-detecting code: We define the decoder as follows

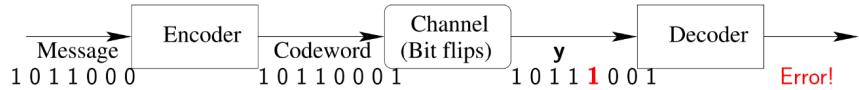


Figure 34.3: Error Detecting Code example

$$c = \begin{cases} y, & \text{if } y \text{ has even parity} \\ \text{ERROR}, & \text{if } y \text{ has odd parity} \end{cases}$$

We now define error to occur when the received message is not the same as the intended message. In this case we assume the channel model that random bit flip of 0 to 1 or 1 to 0 may occur. Now there are various errors that can occur. Note that this code detects an odd number of errors. That is, if odd number of 0s changes to 1s or odd number of 1s changes to 0s. The code can detect an error has occurred but it cannot detect where error has occurred. Also, it cannot detect an error in cases such as when a 0 flips to 1 and a 1 flips to 0 maintaining the parity but the message changes. In this case, it cannot detect an error. Only in the case when it the cases when the parity changes to odd is it can detect error. In all other errors, when the parity remains even it assumes that the codeword is correct (even though there could be cases where the flips occur in such a way that the parity did not change but the message did). For example, in the above case when the codeword 10110001 changes to 10101001, note that the parity is still even but clearly the message has changed). Thus, the decoder designed above can detect error only when y has odd parity, in all other cases, it assumes that no error has occurred. When the decoded detects an error, it sends a message to encoder to resend the message.

These kind of Error-detection codes has applications in Computer network communication protocols, International Standard Book Numbers(ISBN). Network protocols such as TCP/IP use error-detecting codes extensively to determine if data packets sent across a network have been corrupted or not. The usual remedy for

receipt of a corrupted data packet is to request retransmission. ISBN-10 numbers have 10 digits and are of the form x-xxxx-xxxxx-x. The first nine digits in an ISBN record information about the country book: country, published, title, and edition. The tenth digit is a check digit.

34.2.1 Memoryless Binary symmetric channels

Say Alice wants to send a k -bit message to Bob. The message is to be transmitted across a memoryless binary Symmetric Channel. Each bit transmitted across the channel gets flipped with probability p , independently of other bits. Say $k = 10000$ and $p = 0.001$. Say the transmission is uncoded, thus, the rate = 1. The probability that the entire k -bit message is recovered by Bob:

$$P_C = (1 - p)^k = 0.000045$$

for $k = 10000$ and $p = 0.001$.

For an example of a situation in which the chances of making an error can be decreased using a repetition code, suppose that our goal is to communicate a single bit to a hypothetical receiver, and we're able to transmit bits through a so-called binary symmetric channel, which flips each bit sent through it independently with some probability p . That is, with probability $1 - p$ the receiver gets whatever bit was sent through the channel, bit with probability p the bit flips and the receiver gets the opposite bit value.

34.2.2 Classical repetition codes

We'll begin the lesson with classical repetition codes, which form the basis for the 9-qubit Shor code.

Encoding and Decoding

Repetition codes are extremely basic examples of error correcting codes. The idea is that we can protect bits against error by simply repeating each bit some fixed number of times. In particular, let's first consider the 3-bit repetition code. Here we encode one bit into three by repeating the bit three times, so 0 is encoded as 000 and 1 is encoded as 111.

If nothing goes wrong we can obviously distinguish the two possibilities for the original bit from their encodings. The point is that if there was an error and one of the three bits flipped, meaning that a 0 changes into a 1 or a 1 changes to 0, then we

can still figure out what the original bits was by determining which one of the two binary values appear twice. Equivalently, we can decode by computing the majority value (i.e., the binary value that appears most frequently)

$$abc \rightarrow \text{majority}(a, b, c)$$

Of course, if 2 or 3 bits of the encoding flip, then the decoding won't work properly and the wrong bit will be recovered, but if at most 1 of the 3 bits flips, the decoding will be correct. Thus, this code can correct single errors and the rate is $\frac{1}{3}$. This is a typical property of error correcting codes: they may allow for the correction of errors, but only if there aren't too many of them.

So, if we choose not to use the 3 bit repetition code, and simply send whatever bit we have in mind through the channel, the receiver therefore receives the wrong bit with probability p . On the other hand, if we first encode the bit we want to send using the 3 bit repetition code, and then send each of the three bits of the encoding through the channel, then each of them flips independently with probability p . The changes of a bit-flip are now greater because there are now three bits that might flip rather than one - but if at most one bit flips then the receiver will decode correctly. So, an error persists after decoding only if two or more of the bits flip during transmission.

The probability that two bits flip during the transmission is $3p^2(1 - p)$, which is $p^2(1 - p)$ for each of the three choices for the bit that doesn't flip, while the probability that all the three bits flip is p^3 . The total probability of two or three bit flips is therefore

$$1 - P_{b,C} = 3p^2(1 - p) + p^3 = 3p^2 - 2p^3$$

Thus, the probability that an entire k -bit message is decoded correctly is

$$P_C = (P_{b,c})^k \approx 0.97$$

for $k = 10000$ and $p = 0.001$. Note that 30000 coded bits are actually transmitted since the rate is $\frac{1}{3}$.

For values of p smaller than one-half this results in a decrease in the probability that the receiver ends up with the wrong bit. There will still be a chance of an error in this case, but the code decreases the likelihood (For values of p greater than one-half, on the other hand, the code actually increases the likelihood that the receiver gets the wrong bit.) See figure 34.4.

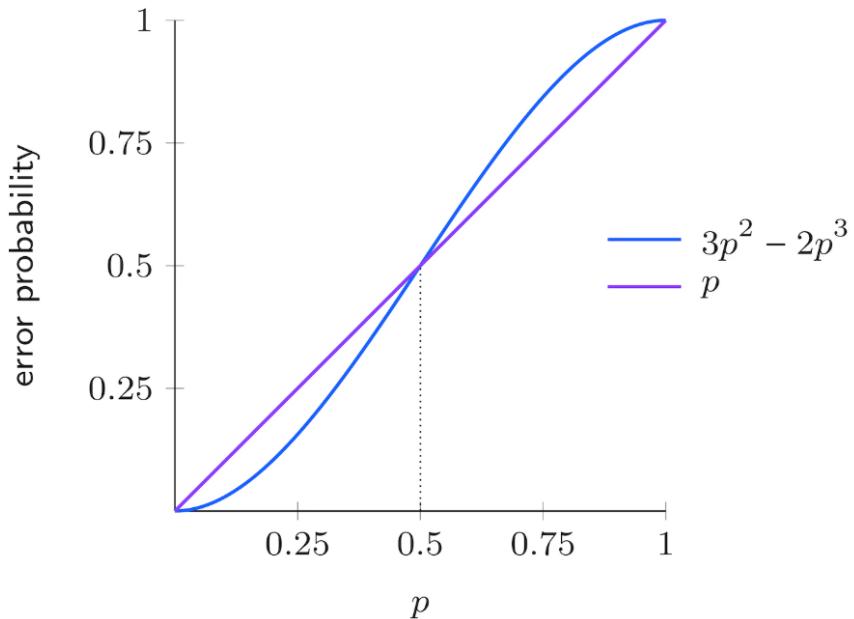


Figure 34.4: Error probability vs p

34.3 Error-Correcting Code

34.3.1 The Length-7 Hamming Code

This code was discovered by Richard hamming in 1950. It encodes a 4 bit message into a 7-bit codeword:

$$u_1 u_2 u_3 U - 4 \rightarrow u_1 u_2 u_3 u_4 p_1 p_2 p_3$$

where $u_i; i = 1, 2, 3$ are information bits and $p_i; i = 1, 2, 3$ are parity bits. Thus, the rate for the code is $\frac{4}{7}$. Consider the following figure 34.5. For each choice of (u_1, u_2, u_3, u_4) , there is a unique choice of (p_1, p_2, p_3) that makes each circle have even number of 1s.

This can be written as follows in the form of equations:

$$\begin{aligned} u_1 + u_2 + u_4 + p_1 &\equiv 0 \pmod{2} \\ u_1 + u_3 + u_4 + p_2 &\equiv 0 \pmod{2} \\ u_2 + u_3 + u_4 + p_3 &\equiv 0 \pmod{2} \end{aligned}$$

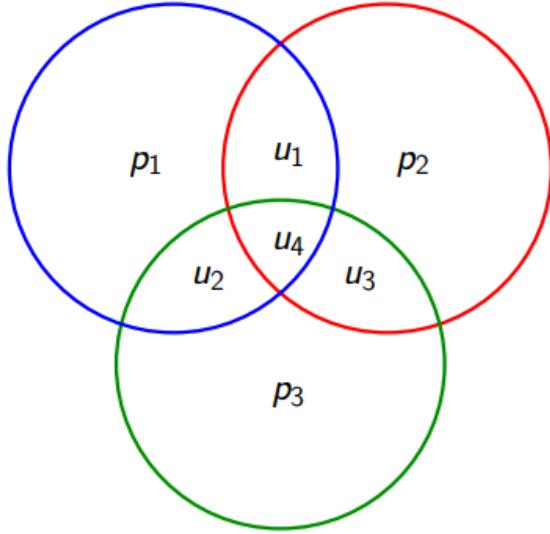


Figure 34.5: Length-7 Hamming Code

For example, if the information bits $u_1 = 1, u_2 = 0, u_3 = 0$ then the codeword in the Hamming code will be 1000110 where $p_1 = 1, p_2 = 0, p_3 = 0$.

Now, consider the following Decoder for Hamming code. Let Alice send information (u_1, u_2, u_3, u_4) which gets encoded to $(u_1, u_2, u_3, u_4, p_1, p_2, p_3)$ in the encoder. Thus, the codeword is $(u_1, u_2, u_3, u_4, p_1, p_2, p_3)$. We assume the channel noise model that flips bits at random ($o \rightarrow 1, 1 \rightarrow 0$). Suppose that Alice sends 1000110, but Bob receives 1100110. Note that the second bit is flipped from 0 to 1. In order to decode we follow the following procedure. Consider the figure 34.6. We use the following rules to decode. Thus, decoder (Bob) uses the following decoding model:

1. Identify circles with wrong parity.
2. Flip bit common to those circles only.

We can clearly, see that circles blue and green are the ones with wrong parity and thus, the bit common only to these two circles is as shown in figure 34.6. Thus, we flip that 1 to 0 and we now have all the circles satisfying the parity. Hence, we have corrected the code and recovered the original message. Like in the case of 3-bit repetition codes based on majority, we can only correct single errors. If there are errors in more than one bit then they cannot be corrected. But there is an interesting case, consider the case when there are errors in two bits. In that case, we can may not be able to correct the errors but we can still detect errors and thus, can request

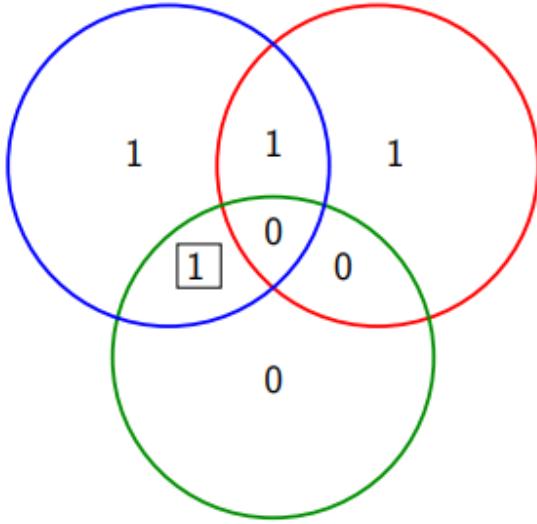


Figure 34.6: 7-bit Hamming Code Example

for retransmission. In the case when the errors are in more than 2 bits we are may not be able to even detect nor correct an error that has occurred.

Now consider we wish to transmit a k -bit message, we first divide the message into 4-bit blocks, encode using a 7-bit codeword, and transmit. Thus, $k/4$ codewords i.re. $7k/4$ coded bits are transmitted. Recall that we can correct one-bit errors. Thus, the probability that the codeword is transmitted correctly or with at most one bit error is

$$Pr[\text{at most one of the 7 bits is flipped}] = (1 - p)^7 + 7p(1 - p)^6$$

Hence, the probability that the entire k -bit message is recovered correctly by Bob is

$$P_C = [(1 - p)^7 + 7p(1 - p)^6]^{k/4} \approx 0.95$$

for $k = 10000$ and $p = 0.001$ where we transmitted $7k/4 = 7 \times 10000/4 = 17500$ coded bits.

Important Note

Shannon's Noisy Channel Coding Theorem

We assume the Channel model to be memoryless binary symmetric channel with crossover probability p . Then the Channel Capacity of the Binary Symmetric Channel is given as:

$$C(p) = 1 + p \log_2 p + (1 - p) \log_2(1 - p)$$

Theorem 34.3.1. *If rate $R < C(p)$ and k is sufficiently large, there exists a code which can encode k message bits into $n = k/R$ coded bits to be transmitted across the channel, such that a decoder at the channel output can recover all k message bits correctly with probability close to 1.*

Note that $C(0.001) \approx 0.9886$. This means that to ensure that a 10000-bit message can be recovered correctly with probability close to 1, it should be enough to transmit about $10000/0.9886 \approx 10150$ coded bits.

The goal of coding theory is to design coding schemes that can approach Shannon's performance guarantees, while still being relatively easy to implement in practice. We are interested in low complexity, and good error-correction capability.

34.4 Linear Codes

Let \mathbb{F}_q denote a finite field of size q . For concreteness, take $q = 2$, i.e. \mathbb{F}_2 is the binary field $\{0, 1\}$ with modulo-2 arithmetic. We will denote $\mathbb{F}_q^n = \{(x_1, x_2, \dots, x_n) : x_i \in \mathbb{F}_q\}$. Clearly, the number of elements, $|\mathbb{F}_q^n| = |\mathbb{F}_q|^n = q^n$.

Definition 34.4.1. A linear code \mathcal{C} of block length n over \mathbb{F}_q is a linear subspace of \mathbb{F}_q^n . For each pair $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}$ and for any $\alpha_1, \alpha_2 \in \mathbb{F}_q$, we also have $\alpha_1 \mathbf{c}_1 + \alpha_2 \mathbf{c}_2 \in \mathcal{C}$.

The dimension of a code \mathcal{C} is the dimension of \mathcal{C} as a linear subspace of \mathbb{F}_q^n over \mathbb{F}_q ; denoted by $\dim(\mathcal{C})$ or $\dim_{\mathbb{F}_q}(\mathcal{C})$.

A $[n, k]$ linear code is a linear code of blocklength n and dimension k .

Proposition 34.4.1. *An $[n, k]$ linear code \mathcal{C} over \mathbb{F}_q has q^k codewords.*

Proof. An $[n, k]$ linear code \mathcal{C} can be uniquely expressed as a linear combination

$$\mathbf{c} = \alpha_1 \mathbf{c}_1 + \alpha_2 \mathbf{c}_2 + \dots + \alpha_k \mathbf{c}_k$$

with $\alpha_j \in \mathbb{F}_q \forall j$. Thus, there is a 1 – 1 correspondence between codewords $\mathbf{c} \in \mathcal{C}$ and k -tuples of coefficients $(\alpha_1, \dots, \alpha_k) \in \mathbb{F}_q^k$. Hence $|\mathcal{C}| = |\mathbb{F}_q^k| = q^k$. \square

The rate of an $[n, k]$ linear code over \mathbb{F}_q is

$$R = \frac{k}{n}$$

Example 34.4.1. The repetition code over $\mathbb{F}_2 : \{00\dots 0, 11\dots 1\}$. This is a linear code over \mathbb{F}_2 with block length n and dimension 1. In other words, this is an $[n, 1]$ binary linear code.

Example 34.4.2. The single parity-check code over \mathbb{F}_2 :

$$\mathcal{C} = \{x_1 x_2 \dots x_n : x_1 + x_2 + \dots + x_n \equiv 0 \pmod{2}\} = \text{nullspace}(H)$$

where $H = [1 \ 1 \ \dots \ 1]$. By the rank-nullity theorem,

$$\dim(\mathcal{C}) = n - \text{rank}(H) = n - 1$$

Thus, \mathcal{C} is an $[n, n - 1]$ binary linear code.

Example 34.4.3. (The Length-7 Hamming code) Recall that a binary word x_1, x_2, \dots, x_7 is in the Hamming code iff

$$\begin{aligned} x_1 + x_2 + x_4 + x_5 &\equiv 0 \pmod{2} \\ x_1 + x_3 + x_4 + x_6 &\equiv 0 \pmod{2} \\ x_2 + x_3 + x_4 + x_7 &\equiv 0 \pmod{2} \end{aligned}$$

rewriting this equations in matrix form (over \mathbb{F}_2) as

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The binary word $x_1 x_2 x_3 x_4 x_5 x_6 x_7$ is the Hamming code \mathcal{C} iff the above equations are satisfied.

In other words, the Hamming code \mathcal{C} is equal to $\text{nullspace } \mathbb{F}_2(H)$. Consequently, by rank-nullity theorem, $\dim(\mathcal{C}) = n - \text{rank}_{\mathbb{F}_2}(H) = 7 - 3 = 4$. Thus, \mathcal{C} is a $[7, 4]$ linear code.

34.4.1 Minimum Distance

Let $\mathbf{c} = (c_1, c_2, \dots, c_n)$ and $\mathbf{c}' \in (c'_1, c'_2, \dots, c'_n)$ be words in \mathbb{F}^n .

Definition 34.4.2. (Hamming Distance) The Hamming distance between \mathbf{c} and \mathbf{c}' is defined to be

$$d(\mathbf{c}, \mathbf{c}') = |\{i : c_i \neq c'_i\}|$$

Definition 34.4.3. (Hamming Weight) The Hamming weight of \mathbf{c} is defined to be

$$w(\mathbf{c}) = d(\mathbf{c}, \mathbf{0}) = |\{i : c_i \neq 0\}|$$

Definition 34.4.4. (Minimum distance) The minimum distance of a linear code \mathcal{C} is

$$d_{min}(\mathcal{C}) = \min_{\mathbf{c}, \mathbf{c}' \in \mathcal{C}: \mathbf{c} \neq \mathbf{c}'} d(\mathbf{c}, \mathbf{c}') = \min_{\mathbf{c} \in \mathcal{C}: \mathbf{c} \neq \mathbf{0}} w(\mathbf{c})$$

An $[n, k, d]$ linear code is an $[n, k]$ linear code with minimum distance d .

Example 34.4.4. The repetition code over $\mathbb{F}_2 : \{00\dots 0, 11\dots, 1\}$ is a $[n, 1, n]$ binary linear code.

Example 34.4.5. The single parity-check code over \mathbb{F}_2 :

$$\mathcal{C} = \{x_1 x_2 \dots x_n : x_1 + x_2 + \dots + x_n \equiv 0 \pmod{2}\}$$

Since there are no codewords of (odd) weight 1, and all binary words of (even) weight are in the code $\dim(\mathcal{C}) = 2$. Thus, this is an $[n, n - 1, 2]$ binary linear code.

Example 34.4.6. (The Length-7 Hamming code) A binary words $x_1 x_2 x_3 x_4 x_5 x_6 x_7$ is in the Hamming code \mathcal{C} iff

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

i.e. $Hx^T = \mathbf{0}$, we can simplify the above expression into

$$x_1 \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + x_2 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + x_3 \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} + x_4 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + x_5 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + x_6 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + x_7 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Now, note that \mathcal{C} has no codewords of weight 1, as no columns of H is $\mathbf{0}$. \mathcal{C} has no codewords of weight 2, as no two columns of H are identical. \mathcal{C} does have codewords of weight 3: e.g., the first three columns of H sum to $\mathbf{0}$ over \mathbb{F}_2 , so 1110000 is in \mathcal{C} . Hence, $d_{\min}(\mathcal{C}) = 3$. Thus, the Hamming code is a $[7, 4, 3]$ binary linear code.

Let \mathcal{C} be a linear code of \mathbb{F} with parity-check matrix H , i.e., $\mathcal{C} = \text{nullspace}_{\mathbb{F}}(H)$.

Theorem 34.4.2. *The minimum distance of \mathcal{C} is equal to the smallest number of columns of H that are linearly dependent over \mathbb{F} . Equivalently, $d_{\min}(\mathcal{C})$ is the largest integer d such that every collection of $d - 1$ columns of H is linearly independent over \mathbb{F} .*

If $\mathcal{C} = \text{nullspace}_{\mathbb{F}}(H)$, then H is called a parity-check matrix for \mathcal{C} .

34.4.2 Error Correction

Given a length n code \mathcal{C} . An error is an even of changing an entry in a codeword. A code words $\mathbf{c} \in \mathcal{C}$ is transmitted, and $\mathbf{y} \in \mathbb{F}^n$ is received. The number of errors that have occurred is $d(\mathbf{y}, \mathbf{c})$.

Definition 34.4.5. A code \mathcal{C} is t -error-correcting if there exists a decoding map $D : \mathbb{F}^n \rightarrow \mathcal{C}$ such that whenever $d(\mathbf{y}, \mathbf{c}) \leq t$, we have $D(\mathbf{y}) = \mathbf{c}$.

Proposition 34.4.3. *A $[n, k, d]$ code is t -error correction for $t < d/2$.*

Proof. A codeword $\mathbf{c} \in \mathcal{C}$ is transmitted, and $\mathbf{y} \in \mathbb{F}^n$ is received. The number of errors that have occurred is $d(\mathbf{y}, \mathbf{c})$. Decode \mathbf{y} to closest codeword: $\hat{\mathbf{c}} = \arg \min_{\mathbf{c}' \in \mathcal{C}} d(\mathbf{y}, \mathbf{c}')$. If $d(\mathbf{y}, \mathbf{c}) < d/2$, then $\hat{\mathbf{c}} = \mathbf{c}$. \square

34.4.3 Erasures

An erasures is an error whose location is known. Usualy represented by a ‘?’ symbol:

$$c_1 c_2 c_3 \dots c_n \longrightarrow \boxed{\text{Channel}} \longrightarrow c_1 ? c_3 ?? c_6 \dots c_n$$

Figure 34.7: Erasure

Proposition 34.4.4. *Let \mathcal{C} be an $[n, k, d]$ code over \mathbb{F} . There is a decode for \mathcal{C} that corrects any occurrence of up to $d - 1$ erasures.*

Proof. Let $\Phi = \mathbb{F} \cup \{?\}$. Consider the decoder defined for each $\mathbf{y} \in \Phi^n$ as

$$\mathcal{D}(\mathbf{y}) = \begin{cases} \mathbf{c} & \text{if } \mathbf{c} \text{ is the unique codeword that agrees with } \mathbf{y} \text{ on all unerased positions} \\ \text{ERROR} & \text{otherwise} \end{cases}$$

Suppose that \mathbf{c} was transmitted and at most $d - 1$ of its coordinates were erased. The received word \mathbf{y} contains at most $d - 1$ ‘?’ symbols, and agrees with \mathbf{c} on all the unerased positions. If there were another $\mathbf{c}' \in \mathcal{C}$ that also agreed with \mathbf{c} on all the unerased positions, then \mathbf{c} and \mathbf{c}' could differ only in those positions where \mathbf{y} has ‘?’ symbols. Then, $d\mathbf{9}\mathbf{c}, \mathbf{c}' \leq d - 1$, which contradicts $d_{min}(\mathcal{C}) = d$. Thus, \mathbf{c} is the unique codeword that agrees with \mathbf{y} in all unerased coordinated, and hence $\mathcal{D}(\mathbf{y}, \mathbf{c})$. \square

Chapter 35

Quantum Noise and Quantum Error-Correcting Codes

Quantum computation has the potential to enable efficient solution to computational tasks for which efficient classical algorithms are not known, and possibly don't exist. There are, however, very significant challenges that will need to be overcome before we can reliably implement the sorts of large-scale quantum computations we hope will one day be possible.

The heart of the matter is that quantum information is extremely fragile - you can literally ruin it just by looking at it. For this reason, to correctly operate, quantum computers need to isolate the quantum information they store from the environment around them to an extreme degree. But at the same time, quantum computers must provide the user with very precise control over this quantum information, including proper initialization, accurate and reliable unitary operations, and the ability to perform measurements so that the result of the computation can be obtained.

There's clearly some tension between these requirements, and in the early days of quantum computing some viewed that the fragility of quantum information, and its susceptibility to both inaccuracies and environmental noise, would ultimately make quantum computing impossible. Today, there's little doubt that building an accurate and reliable large-scale quantum computer is a monumental challenge. But we have a key tool to help us in this endeavour that leads most people who are knowledgeable about the field to be optimistic about large-scale quantum computing one day becoming a reality, and that tool is quantum error correction.

Next we will discuss quantum error correction, with a focus on the fundamentals. In this we'll take a first look at quantum error correction, including the very first quantum error correction code discovered - the 9 qubit Shor code and we will also dis-

cuss a foundational concept in quantum error correction known as the discretization of errors.

35.1 Repetition code for qubits

The 3 bit repetition code is a classical error-correcting code, but we can consider what happens if we try to use it to protect qubits against errors. As we'll see, it's not a very impressive quantum error correcting code: it actually makes some errors more likely. It is, however, the first step towards the Shor code, and it will also serve us well from a pedagogical viewpoint.

35.1.1 Encoding

To be clear, when we refer to the 3-bit repetition code being used for qubits, we have in mind an encoding of a qubit where standard basis states are repeated three times, so that single-qubit state vector is encoded as follows

$$\alpha |0\rangle + \beta |1\rangle \rightarrow \alpha |000\rangle + \beta |111\rangle$$

This encoding is easily implemented by the following quantum circuit in figure 35.1 that makes use of two initialized workspace qubits and two controlled-NOT gates. Notice in particular that this encoding is not the same as repeating the quantum state

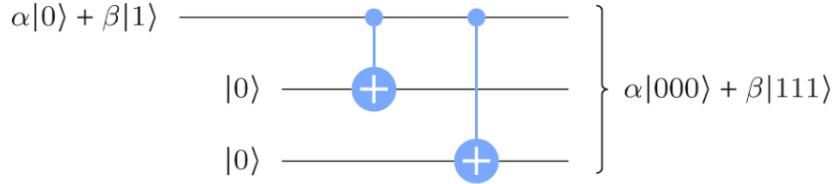


Figure 35.1: 3-bit repetition code circuit

three times, as in a given qubit state vector being encoded as $|\psi\rangle \rightarrow |\psi\rangle |\psi\rangle |\psi\rangle$. Such an encoding cannot be implemented for an unknown quantum state $|\psi\rangle$ by the no cloning theorem.

35.1.2 Bit-Flip error detection

Now suppose that an error takes place after the encoding has been performed. In particular, let's suppose that an X gate, or in other words a bit flip occurs on one of the qubits. For instance, if the middle qubit experiences a bit-flip, the state of the three qubits is transformed into this state:

$$\alpha |010\rangle + \beta |101\rangle$$

Of course this isn't the only sort of error that could occur - and it's also reasonable to question the assumption that an error takes the form of a perfect, unitary operation. We'll return to these issues in the last section of the lesson, and for now we can view an error of this form as being just one possibility (albeit a fundamentally important one) for an error.

We can see clearly from the mathematical expression for the state above that the middle bit is the one that's different inside of each ket - but suppose that we had the three qubits in our possession and didn't know their state. If we suspected that a bit-flip may have occurred, one option to verify that a bit flipped would be to perform a standard basis measurement, which in the case at hand causes us to see 010 or 101 with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. In either case our conclusion would be that the middle bit flipped - but unfortunately the original quantum state $\alpha|0\rangle + \beta|1\rangle$ is now lost. This is the state we're trying to protect, so measuring the standard basis is an unsatisfactory option.

What we can do instead is to use the following quantum circuit, feeding the encoded state into the top three qubits. This circuit in figure 35.2 non-destructively measures the parity of the standard basis states of the top two qubits as well as the bottom two qubits of the three-qubit encoding. Under the assumption that at most one bit flipped, one can easily deduce from the measurement outcomes the location of the bit flip (or the absence of one). In particular, as the following four circuit diagrams 35.3, illustrate the measurement outcome 00 indicates that no bit flip occurred, while the three other possibilities indicate which qubit experienced a bit flip. Crucially, the state of the top three qubits does not collapse in any of the cases, which allows us to correct a bit-flip error if one has occurred - by simply applying it again. The following table 35.1 summarizes the states we obtain from at most one bit flip, the measurement outcomes (which are called the syndrome in the context of error correction), and the correction needed to get back to the original encoding.

Once again, we're only considering the possibility that at most one bit-flip occurred. This wouldn't work correctly if two or three bit-flips occurred and we also haven't considered other possible errors besides bit-flips.

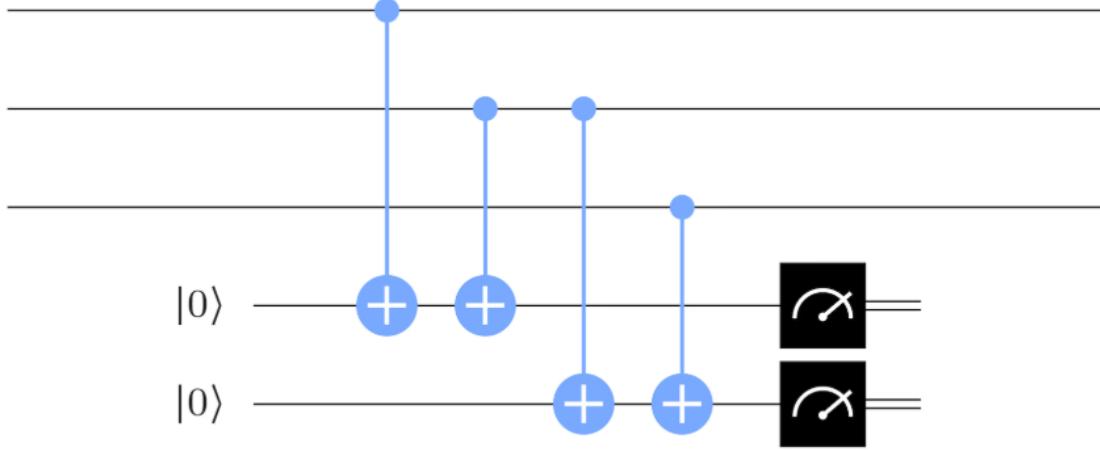


Figure 35.2: Bit-Flip Error correction circuit

State	Syndrome	Correction
$\alpha 000\rangle + \beta 111\rangle$	00	$I \otimes I \otimes I$
$\alpha 100\rangle + \beta 011\rangle$	10	$X \otimes I \otimes I$
$\alpha 010\rangle + \beta 101\rangle$	11	$I \otimes X \otimes I$
$\alpha 001\rangle + \beta 110\rangle$	01	$I \otimes I \otimes X$

Table 35.1: One bit-flip error correction

35.2 Phase-flip errors

In the quantum setting, bit flip errors aren't the only errors we need to worry about. For instance, we also have to worry about phase-flip errors, which are described by Z gates. Along the same lines as bit-flip errors, we can think about phase-flip errors as representing just another possibility of an error that can affect a qubit. However, as we will see in the discretization of errors for quantum error correcting codes, a focus on bit-flip errors and phase-flip errors turns out to be well-justified. Specifically, the ability to correct a bit-flip error, a phase-flip error, or both of these errors simultaneously automatically implies the ability to correct an arbitrary quantum error on a single qubit.

Unfortunately, the 3-bit repetition code doesn't protect against phase flips at all. For instance, suppose that a qubit state $\alpha|0\rangle + \beta|1\rangle$ has been encoded using the

3-bit repetition code, and a phase-flip error occurs on the middle qubit. This results in the state

$$(I \otimes Z \otimes I)(\alpha |000\rangle + \beta |111\rangle) = \alpha |000\rangle - \beta |111\rangle$$

which is exactly the state we would have obtained from encoding the qubit state $\alpha |0\rangle - \beta |1\rangle$. Indeed, a phase-flip error on any one of the three qubits of the encoding has this same effect, which is equivalent to a phase-flip error occurring on the original qubit prior to the encoding. Under the assumption that original quantum state is an unknown state, there's therefore no way to detect that an error has occurred - because it's a perfectly valid encoding of a different qubit state. In particular, running the error detection circuit from before on the state $\alpha |000\rangle - \beta |111\rangle$ is certain to result in the syndrome 00, which wrongly suggests that no errors have occurred.

Meanwhile, there are now three qubits rather than one that could potentially experience phase-flip errors. So, in a situation in which phase-flip errors are assumed to occur independently on each qubit with some nonzero probability p (similarly to a binary symmetric channel except for phase flips rather than bit flips), this code actually increases the likelihood of a phase-flip error after decoding for small values of p . In particular, we'll get a phase-flip error on the original qubit after decoding whenever there are an odd numbers of phase-flip errors on the three qubits of the encoding, which happens with probability $3p(1-p)^2 + p^3$. This value is larger than p when $0 < p < 1/2$. so the code increases the probability of a phase-flip error for values of p in this range.

35.2.1 Modified repetition code for phase-flip errors

We've observed that the 3-bit repetition code is completely oblivious to phase-flip errors, so it doesn't seem to be very helpful for dealing with this sort of error. We can, however, modify the 3-bit repetition code in a simple way so that it does detect phase flip errors. this modification will render the code oblivious to bit-flip errors - but as we'll see in the next section, we can combine together the 3-bit repetition code with this modified versions to obtain the Shor code, which can correct against both bit-flip and phase-flip errors.

Here is the modified version of the encoding circuit from earlier in the lesson, which will now be able to detect phase-flip errors. The modification is very simple: we simply apply a Hadamard gate to each qubit after performing the two controlled-NOT gates.

A Hadamard gate transforms a $|0\rangle$ into a $|+\rangle$ state, and a $|1\rangle$ state into a $|-\rangle$ state, so the next effect is that the single qubit state $\alpha |0\rangle + \beta |1\rangle$ is encoded as

$$\alpha |+++ \rangle + \beta |--- \rangle$$

where $|+++ \rangle = |+\rangle \otimes |+\rangle \otimes |+\rangle$ and $|--- \rangle = |-\rangle \otimes |-\rangle \otimes |-\rangle$.

A phase flip error, or equivalently a Z gate, flips between the state $|+\rangle$ and $|-\rangle$, so this encoding will be useful for detecting (and correcting) phase-flip errors. Specifically, the error-detection circuit from earlier can be modified as follows 35.4. In words, we take the circuit from before and simply put Hadamard gates on the top three qubits at both the beginning and the end. The idea is that the first three Hadamard gate transform $|+\rangle$ and $|-\rangle$ states back into $|0\rangle$ and $|1\rangle$ states, the same parity checks as before takes place, and then the second layer of Hadamard gates transform the states back to $|+\rangle$ and $|-\rangle$ states so that we recover our encoding. For future, reference, let's observe that this phase flip detection circuit can be simplified as follows 35.5. The following four circuit diagram 35.6 describe how our modified version of the 3-bit repetition code, including the encoding step and the error detection step, functions when at most one phase-flip error occurs. The behavior is similar to the ordinary 3-bit repetition code for bit-flips. Here's an analogous table to the one from above 35.2, this time considering the possibility at most one phase-flip error.

State	Syndrome	Correction
$\alpha +++ \rangle + \beta --- \rangle$	00	$I \otimes I \otimes I$
$\alpha --- \rangle + \beta +++ \rangle$	10	$Z \otimes I \otimes I$
$\alpha +-+ \rangle + \beta -+ \rangle$	11	$I \otimes Z \otimes I$
$\alpha ++- \rangle + \beta - - \rangle$	01	$I \otimes I \otimes Z$

Table 35.2: Phase - flip error correction

Unfortunately, this modified version of the 3-bit repetition code can no longer correct bit-flip errors. All is not lost, however. As suggested previously, we'll be able to combine these two codes into one code - the 9-qubit Shor code - that can correct both bit-flip and phase-flip errors (and therefore any error on a single qubit).

35.3 The 9-qubit Shor code

Now we turn to the 9-qubit Shor code, which is a quantum error correcting code obtained by combining together the two codes considered in the previous section: the 3-bit repetition code for qubits, which allows for the correction of a single-bit flip error, and the modified version of that code, which allows for the correction of a single phase-flip error.

35.3.1 Code description

To be precise, the 9-qubit Shor code is the code we obtain by concatenating the two codes from the previous sections. This means that we first apply the other encoding to each of the three qubits used for the first encoding, resulting in 9 qubits in total. While we could apply the two codes in either order in this particular case, we'll make an arbitrary choice to first apply the modified version of the 3-bit repetition code (which detect phase-flip errors), and then we'll encode each of the resulting three qubits independently using the original 3-bit repetition code (which detects bit-flip errors). Here's a circuit diagram representation of this encoding 35.7.

As the figure suggest, we'll think about the 9 qubits of the Shor code as being grouped into three blocks of three qubits, where each block is obtained from the second encoding step (which is the ordinary 3-bit repetition code). The ordinary 3-bit repetition code, which here is applied three times independently, is called the inner code in the context, whereas the outer code is the code used for the first encoding step, which is the modified version of the 3-bit repetition code that detects phase-flip errors.

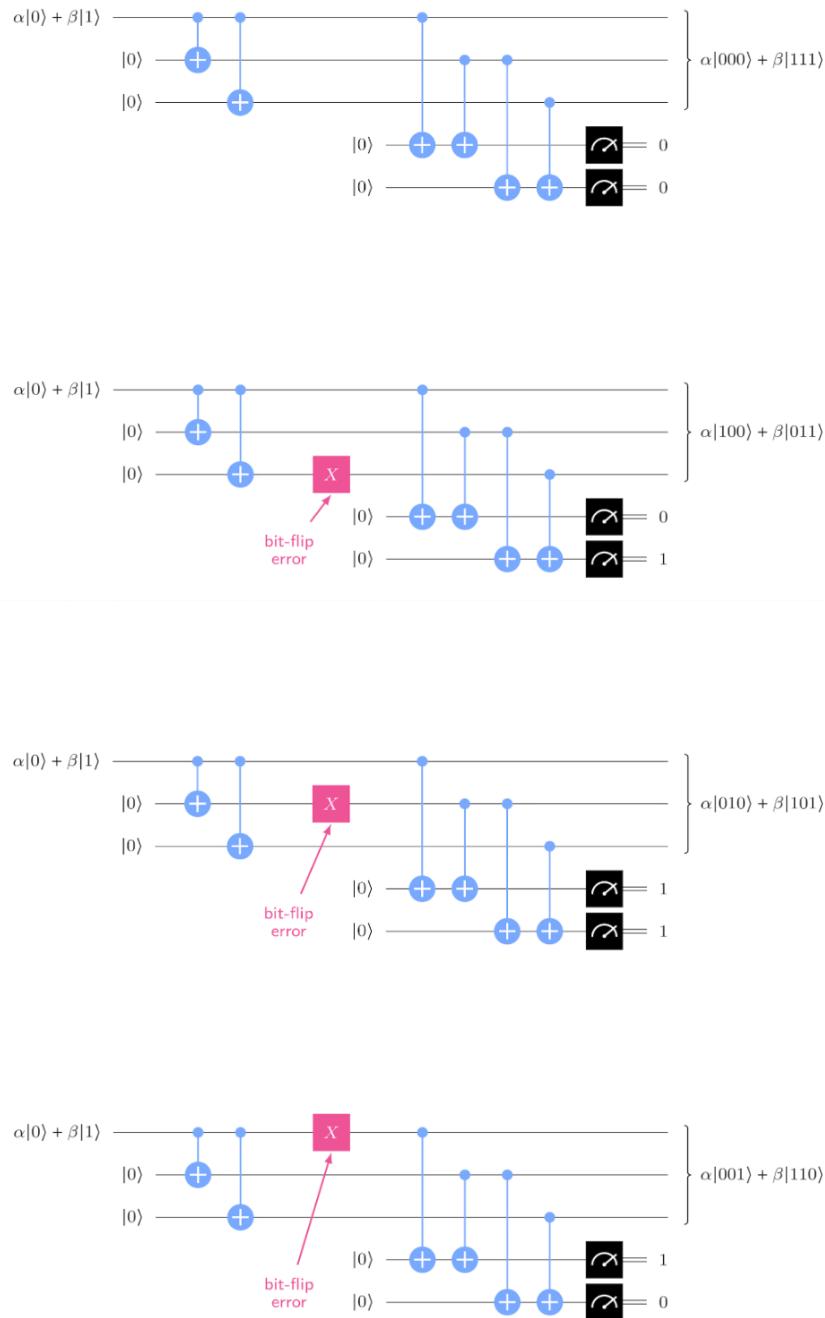


Figure 35.3: Possible Bit-Flip error circuits

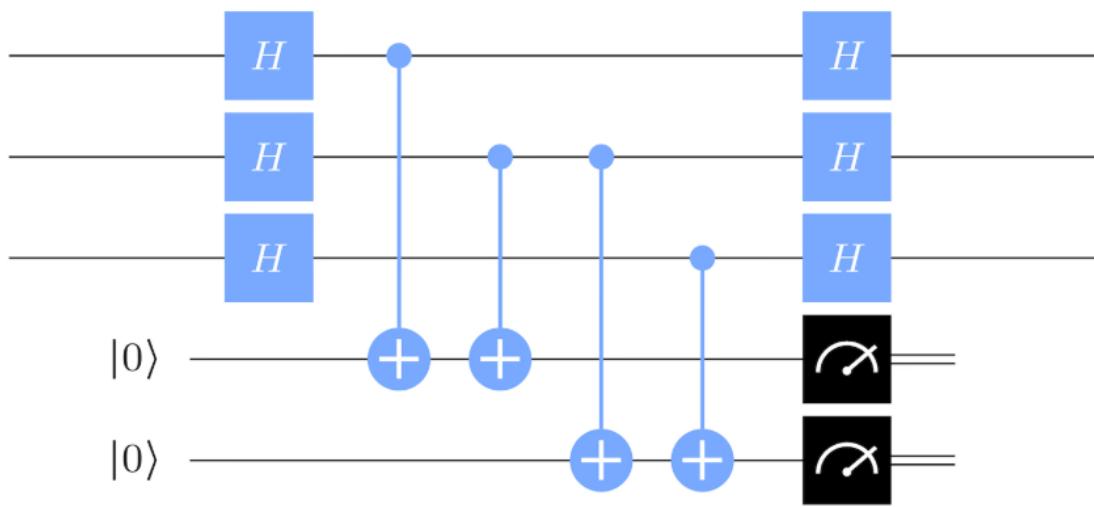


Figure 35.4: Phase-Flip error correction circuit

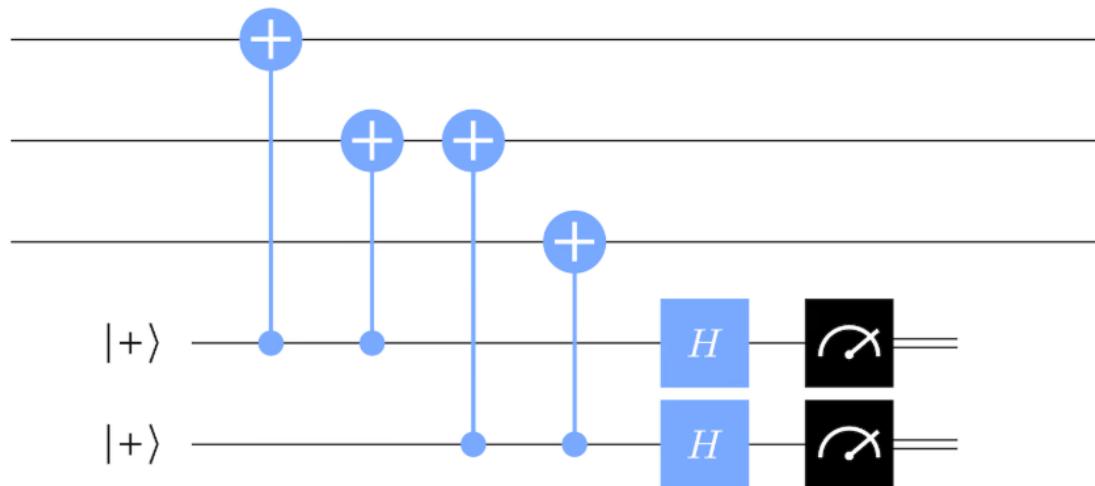


Figure 35.5: Simplified Phase-Flip correction circuit

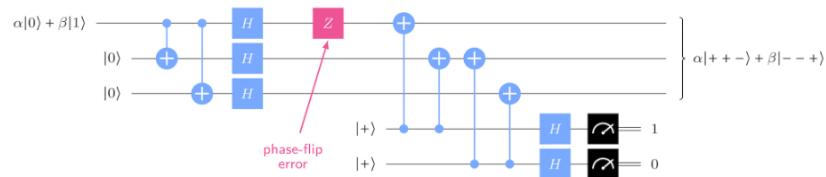
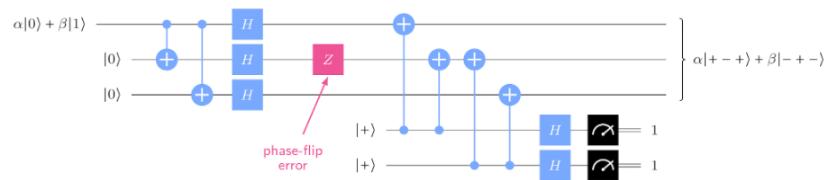
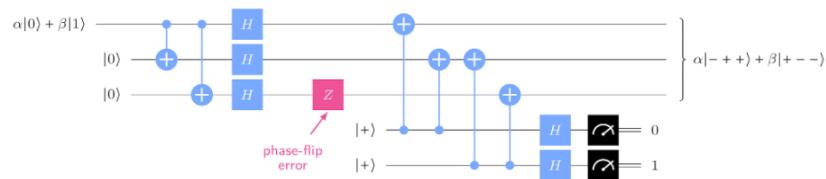
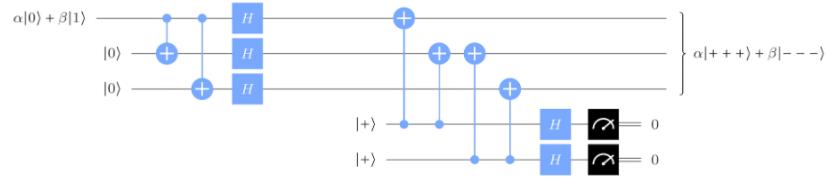


Figure 35.6: Possible Phase-Flip error circuits

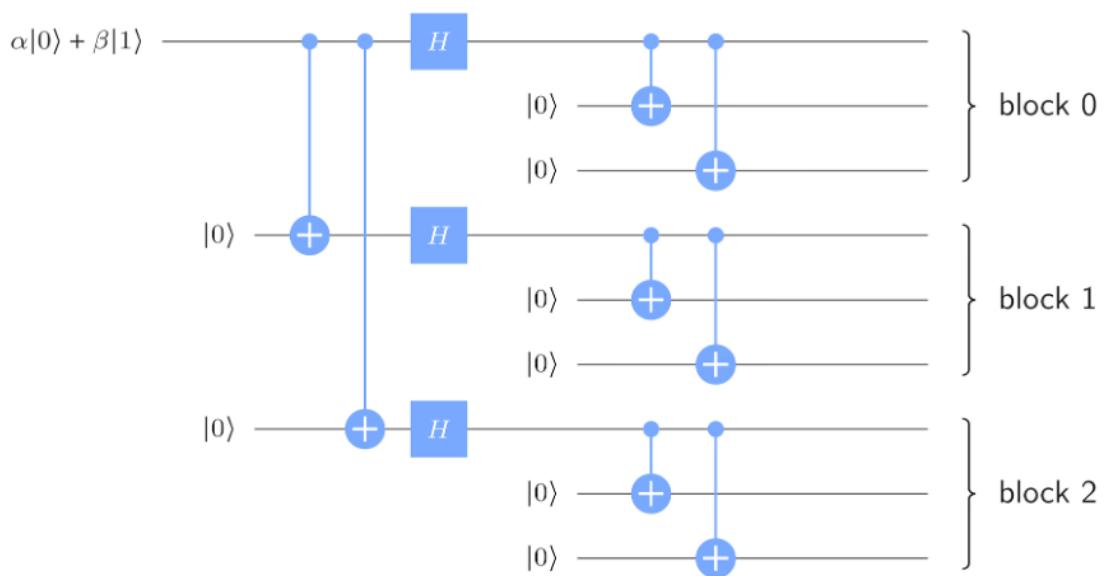


Figure 35.7: Circuit Diagram of 9-qubit Shor code

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 Mechanics 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 2×2 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 $\text{Span}(|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle)$

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.*

Example A.1.11. Linear dependence: example Show that $(1, -1)$, $(1, 2)$ and $(2, 1)$ are linearly dependent. Recall, that the vectors $|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle$ are said to be linearly dependent if the only solution to the equation

$$a_1|v_1\rangle + a_2|v_2\rangle + \dots + a_n|v_n\rangle = 0$$

is trivial solutions $a_1 = a_2 = \dots = a_n = 0$. Thus, writing the equation,

$$\begin{aligned} a_1(1, -1) + a_2(1, 2) + a_3(2, 1) &= (0, 0) \\ (a_1, -a_1) + (a_2, 2a_2) + (2a_3, a_3) &= (0, 0) \\ (a_1 + a_2 + 2a_3, -a_1 + 2a_2 + a_3) &= (0, 0) \\ \implies a_1 + a_2 + 2a_3 &= 0 \\ \implies -a_1 + 2a_2 + a_3 &= 0 \end{aligned}$$

Thus, we have two independent equations and three variables hence, infinite solutions exist. Thus, along with trivial many other non-trivial solutions exists. Hence the vectors are linearly dependent. We can also observe that, putting $a_1 = 1, a_2 = 1, a_3 = -1$, we get,

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Thus, recall that if the finite set S contains zero vector $|0\rangle$ then the set S is linearly dependent. Thus, clearly the set shown here contains the zero vector, hence, its linearly dependent.

We can also apply the theorem that for any vector space V with $\dim(V) = n$. Any set of vectors $m > n$ in V will be linearly dependent. Here, $V = \mathbb{R}^2, n = 2, m = 3$.

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. $\text{Span}(|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle) = V$

Example A.1.12. 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):

$$|v\rangle = \sum_{i=1}^n v_i |i\rangle, |w\rangle = \sum_{i=1}^n w_i |i\rangle$$

$$|v\rangle + |w\rangle = \sum_{i=1}^n (v_i + w_i) |i\rangle$$

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.1.13. 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.

Example A.1.14. Verify that $\langle \cdot | \cdot \rangle$ defined is an inner product on \mathbb{C}^n .

Recall that on \mathbb{C}^n was defined as:

$$\langle \cdot | \cdot \rangle = \sum_i y_i^* z_i = [y_1^* \ y_2^* \ \dots \ y_n^*] \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix}$$

Furthermore, recall the three conditions for the inner product, $\langle \cdot | \cdot \rangle : V \times V \rightarrow \mathbb{C}$ to be considered an inner product:

1. $\langle \cdot | \cdot \rangle$ is linear in the second argument.
2. $\langle v | w \rangle = \langle w | v \rangle^*$
3. $\langle v | v \rangle \geq 0$ with equality if and only if $|v\rangle = 0$.

We check that $\langle \cdot | \cdot \rangle : \mathbb{C}^n \times \mathbb{C}^n \rightarrow \mathbb{C}$ satisfies he three conditions:

1.

$$\begin{aligned} \langle y | \sum_i \lambda_i z_i \rangle &= \sum_i y_i^* [\text{sum}_k \lambda_k z_{ik}] \\ &= \sum_k \lambda_k \sum_i y_i^* z_{ik} \\ &= \sum_k \lambda_k \langle y | z_i \rangle \end{aligned}$$

2.

$$\begin{aligned} \langle y | z \rangle &= \sum_i y_i^* z_i \\ &= \sum_i (y_i z_i^*)^* \\ &= \left(\sum_i z_i^* y_i \right)^* \\ &= \langle z | y \rangle^* \end{aligned}$$

3. Observe, $\mathbf{0} = (0 \ \dots \ 0)$

$$\langle 0 | 0 \rangle = \sum_i 0 \cdot 0 = 0$$

For $y \neq 0$ we have that at least one y_i is non zero, and hence

$$\langle y|y \rangle = \sum_i y_i^* y_i = \sum_i |y_i|^2 \geq 0$$

which proves the claim.

Example A.1.15. Show that any inner product is conjugate-linear in the first argument

$$\left\langle \left(\sum_i \lambda_i |w_i\rangle \right) |v\rangle \right.$$

This property is called **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:

Using property 1: Conjugate Symmetry we get,

$$\left\langle \left(\sum_i \lambda_i |w_i\rangle \right) |v\rangle = \langle v | \left(\sum_i \lambda_i |w_i\rangle \right) \right\rangle^*$$

Using Property 2: Linearity in second argument we get,

$$= \left(\sum_i \lambda_i \langle v | w_i \rangle \right)^*$$

using the properties of Complex Conjugate we get,

$$= \sum_i \lambda_i^* \langle w_i | v \rangle$$

where λ_i^* is the complex conjugate of λ_i .

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.1.16. 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.1.17. Consider a ket $|v\rangle = \frac{1}{2} \begin{pmatrix} 1 + i \\ 1 - i \end{pmatrix} \in \mathbb{C}^2$. The corresponding bra is given by $\langle v| = \frac{1}{2} (1 - i \quad 1 + i)$, and the length of $|v\rangle$ is

$$\sqrt{\langle v|v\rangle} = \sqrt{\frac{1}{4} 2(1 + i)(1 - i)} = \sqrt{\frac{1}{2}(1 + i - i - i^2)} = \sqrt{\frac{1}{2} 2} = 1$$

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.1.18. 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 $|v_i| \geq 0$).

Example A.1.19. Verify that $|w\rangle = (1, 1)$ and $|v\rangle = (1, -1)$ are orthogonal. What are the normalized forms of these vectors?

$$\langle w|v \rangle = 1 \cdot 1 + 1 \cdot -1 = 1 + (-1) = 0$$

Thus, clearly since $\langle v|w \rangle = 0$ they are orthogonal. Now, $\| |v\rangle \| = \sqrt{\langle v|v \rangle} = \sqrt{1^2 + 1^2} = \sqrt{2}$ and $\| |w\rangle \| = \sqrt{\langle w|w \rangle} = \sqrt{1^2 + 1^2} = \sqrt{2}$. Thus, their normalized forms are: $|v\rangle / \| |v\rangle \| = \frac{1}{\sqrt{2}}(1, 1)$ and $|w\rangle / \| |w\rangle \| = \frac{1}{\sqrt{2}}(1, -1)$.

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.1.20. 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

$$|v\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |w\rangle = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

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

$$\| |u_{k+1}\rangle \| = \left\| \frac{|v_{k+1}\rangle}{\|v_{k+1}\|} \right\| = \frac{\langle v_{k+1}|}{\|v_{k+1}\|} \frac{|v_{k+1}\rangle}{\|v_{k+1}\|} = \frac{\langle v_{k+1}|v_{k+1}\rangle}{\|v_{k+1}\|^2} = \frac{\|v_{k+1}\|^2}{\|v_{k+1}\|^2} = 1$$

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 Triangle 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,

$$\begin{aligned} |Z\rangle &= |V\rangle - \frac{\langle W|V\rangle}{|W|^2} |W\rangle \\ \langle Z| &= \langle V| - \frac{\langle V|W\rangle}{|W|^2} \langle W| \end{aligned}$$

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,

$$\begin{aligned} |\langle V|V\rangle| &\geq \frac{\langle V|W\rangle \langle W|V\rangle}{|W|^2} \\ \langle V|V\rangle \langle W|W\rangle &\geq |\langle V|W\rangle|^2 \\ |\langle V|W\rangle|^2 &\leq \langle V|V\rangle \langle W|W\rangle \\ |\langle V|W\rangle| &\leq \sqrt{\langle V|V\rangle \langle W|W\rangle} \end{aligned}$$

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

$$\langle Z|Z\rangle = |Z|^2 = \langle V| |V\rangle + \langle W| |W\rangle + \langle V| |W\rangle + \langle W| |V\rangle$$

$$|V + W|^2 = |V|^2 + |W|^2 + 2\text{Re}(\langle V|W\rangle)$$

Now using, $\text{Re}(\langle V|W\rangle) \leq |\langle V|W\rangle|$, we get

$$|V + W|^2 \leq |V|^2 + |W|^2 + 2|\langle V|W\rangle|$$

Using Schwarz Inequality, we get

$$|V + W|^2 \leq |V|^2 + |W|^2 + 2|V||W|$$

$$|V + W|^2 \leq (|V| + |W|)^2$$

$$|V + W| \leq |V| + |W|$$

Hence, proved. □

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 $|1'\rangle, |2'\rangle, \dots, |m'\rangle$ and thus, the vector $|v\rangle$ to $|v'\rangle$ as follows:

$$A(|v\rangle) = A\left(\sum_{i=1}^n v_i |i\rangle\right) = \sum_{i=1}^n v_i A(|i\rangle) = \sum_{i=1}^n v_i |i'\rangle = |v'\rangle$$

This can be explained as follows:

$$\begin{aligned} A|v\rangle &= \sum_{i=1}^n Av_i |i\rangle \\ \langle j|A|v\rangle &= \sum_{i=1}^n v_i \langle j|A|i\rangle \end{aligned}$$

For each $\{|i\rangle\}_{i=1}^n$ and each $\{|j'\rangle\}_{j=1}^m$. Thus, the inner product with one of the basis vectors of the output basis results in as shown above. This can be written in the matrix form as follows:

$$A|v\rangle = \begin{bmatrix} \langle 1'|A|1\rangle & \langle 1'|A|2\rangle & \dots & \langle 1'|A|n\rangle \\ \langle 2'|A|1\rangle & \langle 2'|A|2\rangle & \dots & \langle 2'|A|n\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle m'|A|1\rangle & \langle m'|A|2\rangle & \dots & \langle m'|A|n\rangle \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

where the input basis is $|1\rangle, |2\rangle, \dots, |n\rangle$ and the output basis is $|1'\rangle, |2'\rangle, \dots, |m'\rangle$. Thus, we have a matrix representation of A in the given input $|1\rangle, |2\rangle, \dots, |n\rangle$ and output basis $|1'\rangle, |2'\rangle, \dots, |m'\rangle$.

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.1.21. (Matrix representations: example) 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 output basis, we assume that the input and output 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: Identifying $\text{ket}0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. 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 output basis. Note that to make the connection between matrices and linear operators one must specify a set of input and output 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.

Important Note

Remark. If we choose the output and the input basis to be different we can even represent A as an identity matrix. Specifically, if the input basis is chosen as $\{|0\rangle, |1\rangle\}$ and the output basis as $\{|1\rangle, |0\rangle\}$, the matrix representation of A looks like:

$$\begin{bmatrix} \langle 1|A|0\rangle & \langle 1|A|1\rangle \\ \langle 0|A|0\rangle & \langle 0|A|1\rangle \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Note that this implies that even the order of input and output basis matters!!

Example A.1.22. (Basis Changes) Suppose A' and A'' are matrix representation of an operator A on a vector space V with respect to two different orthonormal basis $|v_i\rangle$ and $|w_i\rangle$. Then the elements of A' and A'' are $A'_{ij} = \langle v_i | A | v_j \rangle$ and $A''_{ij} = \langle w_i | A | w_j \rangle$. Characterize the relationship between A' and A'' .

$$\begin{aligned} A'_{ij} &= \langle v_i | A | v_j \rangle = \langle v_i | IAI | v_j \rangle = \langle v_i | \left(\sum_{i'} |w_{i'}\rangle \langle w_{i'}| \right) A \left(\sum_{j'} |w_{j'}\rangle \langle w_{j'}| \right) |v_j \rangle \\ &= \sum_{i'} \sum_{j'} \langle v_i | w_{i'} \rangle \langle w_{i'} | A | w_{j'} \rangle \langle w_{j'} | v_j \rangle \\ &= \sum_{i'} \sum_{j'} \langle v_i | w_{i'} \rangle A''_{ij} \langle w_{j'} | v_j \rangle \end{aligned}$$

Example A.1.23. (Matrix representation for identity) Show that the identity operator on a vector space V has a matrix representation which is one along the diagonal and zero everywhere else, if the matrix representation is taken with respect to the same output and input bases. This matrix is known as the identity matrix.

Let the basis for the vector space V be $|v_i\rangle$ then the identity operator is such that $I |v_i\rangle = |v_i\rangle$. Let $A : V \rightarrow V$ be a linear operator, and let its matrix representation

taken to be with respect to $|v_i\rangle$ as the input and output bases. Thus, its matrix representation with respect to the same input and output bases is given as

$$\begin{aligned} I_{ij} &= \langle v_i | I | v_j \rangle \\ &= \langle v_i | v_j \rangle \\ I_{ij} &= \delta_{ij} \end{aligned}$$

Thus, it is one along the diagonal where $i = j$ and 0 every where else where $i \neq j$.

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 output bases. We also use the convention that if the input and output spaces for a linear operator are the same, then the input and output 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 δ_{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.

$$\mathbb{P}_i = |i\rangle\langle i| = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix} (0 \quad \dots \quad 0 \quad 1 \quad 0 \quad \dots \quad 0) = \begin{pmatrix} 0 & \dots & 0 & 0 \\ \vdots & 1 & \dots & 0 \\ 0 & \dots & 0 & 0 \end{pmatrix}$$

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. Suppose W is a k -dimensional vector subspace of the d -dimensional vector space V . using the gram-Schmidt procedure it is possible to construct an orthonormal basis $|1\rangle, \dots, |d\rangle$ for V such that $|1\rangle, \dots, |k\rangle$ is an orthonormal basis for W . By definition,

$$P = \sum_{i=1}^k |i\rangle\langle i|$$

is the projector onto the subspace W . It is easy to check that this definition is independent of the orthonormal basis $|1\rangle, \dots, |k\rangle$ used for W . From the definition, it can be shown that $|v\rangle\langle v|$ is hermitian for any vector $|v\rangle$, so P is Hermitian, $P^\dagger = P$. We will often refer to the vector space P , as shorthand for vector space onto which P is a projector. The orthogonal complement of P is the operator $Q = I - P$. It is easy to see that Q is a projector onto the vector space spanned by $|k+1\rangle, \dots, |d\rangle$, which we also refer to as the orthogonal complement of P , and may denote by Q .

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 .

Example A.1.24. (Matrix representation of operator products) Suppose A is a linear operator from vector space V to vector space W , and B is a linear operator from vector space W to vector space to vector space X . Let $|v_i\rangle, |w_j\rangle$, and $|x_k\rangle$ be bases for the vector space V, W and X , respectively. Show that the matrix

representation for the linear transformations BA is the matrix product of the of the matrix representations of B and A , with respect to the appropriate bases.

Recall that the matrix-vector multiplication choosing the input bases itself as the vector, along with the projection on the output bases can be written as

$$\begin{aligned} A|v_i\rangle &= \sum_j \langle w_j | A | v_i \rangle |w_j\rangle & B|w_j\rangle &= \sum_k \langle x_k | B | w_j \rangle |x_k\rangle \\ BA|v_i\rangle &= B \left(\sum_j \langle w_j | A | v_i \rangle |w_j\rangle \right) \\ &= \sum_j \langle w_j | A | v_i \rangle (B|w_j\rangle) \\ &= \sum_j \langle w_j | A | v_i \rangle \left(\sum_k \langle x_k | B | w_j \rangle |x_k\rangle \right) \\ &= \sum_j \sum_k \langle w_j | A | v_i \rangle \langle x_k | B | w_j \rangle |x_k\rangle \\ &= \sum_{jk} A_{ji} B_{kj} |x_k\rangle \\ &= \sum_{jk} B_{kj} A_{ji} |x_k\rangle \\ &= \sum_k \left(\sum_j B_{kj} A_{ji} \right) |x_k\rangle \\ BA|v_i\rangle &= \sum_k (BA)_{ki} |x_k\rangle \end{aligned}$$

which shows that the matrix representation of BA is indeed the matrix product of the representations of B and A .

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.

Example A.1.25. Suppose $|v_i\rangle$ is an orthonormal basis for an inner product space V. What is the matrix representation for the operator $|v_j\rangle\langle v_k|$, with respect to the $|v_i\rangle$ basis.

The l, m element of the matrix will be

$$\begin{aligned} A_{lm} &= \langle v_l | v_j \rangle \langle v_k | v_m \rangle \\ &= \delta_{lj} \delta_{km} \end{aligned}$$

Thus, the matrix will be 1 at the j, m entry of the matrix and 0 everywhere else.

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.1.26.

$$\alpha_1 |V_1\rangle = \alpha_2 |V_2\rangle + \alpha_3 |v_3\rangle \langle V_4 | V_5 \rangle + \alpha_4 AB |V_6\rangle$$

It's adjoint will be given as:

$$\alpha_1^* \langle V_1 | = \alpha_2^* \langle V_2 | + \alpha_3^* \langle v_3 | \langle V_5 | V_4 \rangle + \alpha_4^* \langle V_6 | B^\dagger A^\dagger$$

Anti-linearity of the adjoint The adjoint operation is anti-linear

$$\left(\sum_i a_i A_i \right)^\dagger = \sum_i a_i^* A_i^\dagger$$

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 (or Self-Adjoint 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 or zero.
- 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. In other words, 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.

Example A.1.27. If $|w\rangle$ and $|v\rangle$ are any two vectors, show that $(|w\rangle\langle v|)^\dagger = |v\rangle\langle w|$.

We observe that:

$$\begin{aligned}\langle(|w\rangle\langle v|)^\dagger|x\rangle|y\rangle &= \langle x|\langle v|y\rangle|w\rangle \\ &= \langle x|w\rangle\langle v|y\rangle \\ &= \langle\langle x|w\rangle^*|v\rangle|y\rangle \\ &= \langle\langle w|x\rangle|v\rangle|y\rangle \\ &= \langle(|v\rangle\langle w|)|x\rangle|y\rangle\end{aligned}$$

Where in the third to last equality we use the conjugate linearity in the first argument and in the second-to-last equality we use that $\langle a|b\rangle = \langle b|a\rangle^*$. Comparing the first and last expressions, we conclude that $(|w\rangle\langle v|)^\dagger = |v\rangle\langle w|$.

Example A.1.28. Show that the adjoint operator is anti-linear,

$$\left(\sum_i a_i A_i\right)^\dagger = \sum_i a_i^* A_i^\dagger$$

We observe that:

$$\begin{aligned}\langle\left(\sum_i a_i A_i\right)^\dagger|a\rangle|b\rangle &= \langle a|\sum_i a_i A_i|b\rangle = \sum_i a_i \langle a|A_i|b\rangle \\ &= \sum_i a_i \langle A_i^\dagger|a\rangle|b\rangle \\ &= \langle\sum_i a_i^* A_i^\dagger|a\rangle|b\rangle\end{aligned}$$

where we invoke the definition of the adjoint in the first and third equalities, the linearity in the second argument in the second equality, and the conjugate linearity in the first argument in the last equality. The claim is proven by comparing the first and last expressions.

Example A.1.29. Show that $(A^\dagger)^\dagger = A$.

Applying the definition of the Adjoint twice (and using the conjugate symmetry of the inner product) we have that:

$$\langle(A^\dagger)^\dagger|a\rangle|b\rangle = \langle a|A^\dagger|b\rangle = \langle A^\dagger|b\rangle|a\rangle^* = \langle b|A|a\rangle^* = (\langle A|a\rangle|b\rangle^*)^* = \langle A|a\rangle|b\rangle$$

The claim follows by comparison of the first and last expressions.

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_1U_2 is also unitary.

$$(U_1U_2)^\dagger U_1U_2 = U_2^\dagger U_1^\dagger U_1U_2 = U_2^\dagger U_2 = I$$

- 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 Unitary 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 Diagonalizable: 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 Λ 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 Λ 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 :

$$\begin{aligned} A_H &= V\Lambda_H V^\dagger \\ A_{SH} &= V\Lambda_{SH} V^\dagger \end{aligned}$$

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, Λ_H and Λ_{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 Λ 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.

Example A.1.30. Show that a normal matrix is Hermitian if and only if it has real eigenvalues.

Let A be a Normal and Hermitian matrix. Then, it has a diagonal representation $A = \lambda_i |i\rangle \langle i|$ where $|i\rangle$ is an orthonormal basis for V and each $|i\rangle$ is an eigenvector of A with eigenvalue λ_i . by the Hermicity of A , we have that $A = A^\dagger$. Therefore, we have that

$$A^\dagger = \left(\sum_i \lambda_i |i\rangle \langle i| \right)^\dagger = \sum_i \lambda_i^* |i\rangle \langle i| = A = \sum_i \lambda_i |i\rangle \langle i|$$

where we used the results previously proved. Comparing the third and last expressions, we have that $\lambda_i = \lambda_i^*$ and hence the eigenvalues are real.

Let A be a normal matrix with real eigenvalues. Then, A has diagonal representation $A = \sum_i \lambda_i |i\rangle\langle i|$ where λ_i are all real. We therefore have that:

$$A^\dagger = \left(\sum_i \lambda_i |i\rangle\langle i| \right)^\dagger = \lambda_i^* |i\rangle\langle i| = \sum_i |i\rangle\langle i| = A$$

where in the third equality we use that $\lambda_i = \lambda_i^*$. We conclude that A is Hermitian.

A.1.8 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, if $\langle v|A|v\rangle > 0$, for all $|v\rangle \neq 0$, then we say A is positive definite. i.e. $\langle v|A|v\rangle \geq 0$.

Properties of Positive Operators:

- A positive operator is necessarily Hermitian and hence by spectral decomposition can be written as $\sum_i \lambda_i |\lambda_i\rangle\langle\lambda_i|$, with non-negative eigenvalue λ_i .

Proof: Recall that any operator A can be written as sum of two Hermitian operators B , and C as

$$A = B + \iota C$$

where $B = \frac{A+A^\dagger}{2}$ is Hermitian and $C = -\iota\frac{A-A^\dagger}{2}$ is also Hermitian. Now since A is a positive operator $\Rightarrow \langle v|A|v\rangle > 0$, thus,

$$\begin{aligned} \langle v|A|v\rangle &= \langle v|B|v\rangle + \iota\langle v|C|v\rangle > 0 \\ &\Rightarrow \langle v|C|v\rangle = 0 \quad \forall |v\rangle \neq 0 \\ &\Rightarrow C = 0 \\ &\Rightarrow -\iota\frac{A-A^\dagger}{2} = 0 \\ &\Rightarrow A = A^\dagger \end{aligned}$$

Thus, a positive operator necessarily implies 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.1.9 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 λ 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 λ (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 λ 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 $A - \lambda I$.

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

where \det is the determinant function for matrices. This, results in a characteristic equation of the form:

$$P_A(\lambda) = \det(A - \lambda I) = \lambda^n - \beta_1\lambda^{n-1} + \dots + (-1)^r\beta_r\lambda^{n-r} + \dots + (-1)^n\beta_n = 0$$

The solutions of the characteristic equations are the eigenvalues of the operator A . By fundamental theorem of algebra, every polynomial has at least one complex root, so every operator A has at least one eigenvalues, and a corresponding eigenvector. The eigenspace corresponding to an eigenvalue v is the set of vectors which have eigenvalue v . It is the vector subspace of the vector space on which A acts. When an eigenspace is more than one dimensional we say that it is degenerate. For example,

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

In general, we represent eigen value as ω and the corresponding eigen vector as $|\omega\rangle$.

Example A.1.31. For the identity 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.1.32. 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, λ 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 either $\lambda = 0$ 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.1.33. Show that any projector P satisfies the equation $P^2 = P$.

Let $|1\rangle, \dots, |k\rangle$ be an orthonormal basis for the subspace W of V . Then, using the definition of the projector onto W , we have that:

$$P^2 = P \cdots P = \left(\sum_{i=1}^k |i\rangle \langle i| \right) \left(\sum_{i'=1}^k |i'\rangle \langle i'| \right) = \sum_{i=1}^k \sum_{i'=1}^k |i\rangle \delta_{ii'} \langle i'| = \sum_{i=1}^k |i\rangle \langle i| = P$$

where in the fourth/fifth equality we use the orthonormality of the basis to collapse the double sum.

Example A.1.34. Consider the Rotation operator $R(\vec{\theta})$ which rotates any vector by an angle $\frac{\vec{\theta}}{|\theta|}$ 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 Similar matrices are same.
- Eigen Values of a Unitary matrix is a complex number with unit modulus and the Eigen vectors are mutually orthogonal (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 $u_i u_i^* = |u_i|^2 = 1$ i.e. unit modulus eigen values i.e. $e^{i\theta}$ for some real θ . If $i \neq j$ then $\langle u_j | u_i\rangle = 0$ i.e. the eigen vectors are orthogonal.

Unitary matrix has complex eigen values of unit modulus and orthogonal eigen vectors.

- If λ 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 are 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 λ_i and λ_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 λ is called the Geometric multiplicity. Suppose λ_i has geometric multiplicity k $\implies GM(\lambda_i) = k$ means that k linearly independent eigen vectors exist corresponding to eigen value λ . These vectors form a vector space which is called Eigen Space. In the eigen space all the vectors are only scaled by λ_i and not rotated.

Definition A.1.27. Algebraic Multiplicity: The multiplicity of λ as a root of the characteristic equation $P_A(x) = 0$ is called the algebraic multiplicity. Suppose λ_i has algebraic multiplicity k $\implies AM(\lambda_i) = k$ means that λ_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 λ 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 λ_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 λ_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 λ . 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 λ 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 λ 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 λ . 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 λ . 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 λ . Thus, λ 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

A.1.10 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 λ_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, Λ is a diagonal matrix of corresponding eigen values.

Example A.1.35. Prove the the matrix

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

is not diagonalizable.

A matrix is said to be not diagonalizable if it is a defective matrix i.e. $AM(\lambda_i) \neq GM(\lambda_i)$. The eigenvalues of the matrix are $\lambda = 1, 1$ since its a lower triangular matrix. The corresponding eigenvector is:

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

Thus, the Geometric Multiplicity corresponding to $\lambda = 1$ is $GM(\lambda = 1) = 1$ and the Algebraic Multiplicity is $AM(\lambda = 1) = 2$ which is not equal to the geometric multiplicity and hence the matrix is a defective matrix this, non-diagonalizable.

A.1.11 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

- $AB = \frac{[A, B] + \{A, B\}}{2}$ Verify that

$$AB = \frac{[A, B] + \{A, B\}}{2}$$

By algebraic manipulation we obtain:

$$AB = \frac{AB + BA}{2} + \frac{AB - BA}{2} = \frac{[A, B] + \{A, B\}}{2}$$

- if $[A, B] = 0, \{A, B\} = 0$ and A is invertible then $B = 0$.

We have that:

$$[A, B] = AB - BA = 0$$

$$\{A, B\} = AB + BA = 0$$

Adding the first line to the second we have:

$$2AB = 0 \implies AB = 0$$

A^{-1} exists by the invertibility of A, so multiplying by A^{-1} on the left we have:

$$A^{-1}AB = 0 \implies B = 0$$

- $[A, B]^\dagger = [B^\dagger, A^\dagger]$ Using the properties of the adjoint, we have:

$$[A, B]^\dagger = (AB - BA)^\dagger = (AB)^\dagger - (BA)^\dagger = B^\dagger A^\dagger - A^\dagger B^\dagger = [B^\dagger, A^\dagger]$$

- $[A, B] = -[B, A]$ By the definition of the commutator

$$[A, B] = AB - BA = -(BA - AB) = -[B, A]$$

- If A and B are Hermitian then $\iota[A, B]$ is Hermitian.

Suppose A, B are Hermitian. using the results, we have:

$$(\iota[A, B])^\dagger = -\iota([A, B])^\dagger = -\iota[B^\dagger, A^\dagger] = \iota[A^\dagger, B^\dagger] = \iota[A, B]$$

A.1.12 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.

A.1.13 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. Note that it is a function from complex numbers to 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.

It is possible to define a corresponding matrix function on normal matrices (or some subclass, such as the Hermitian matrices) by the following construction. Let $A = \sum_a a |a\rangle\langle a|$ be a spectral decomposition of a normal operator A . Define $f(A) = \sum_a f(a) |a\rangle\langle a|$. A little thought shows that $f(A)$ is uniquely defined. This procedure can be used, for example, to define the square root of a positive operator, the logarithm of a positive-definite operator, or the exponential of a normal operator. For example,

$$\exp(\theta Z) = \begin{bmatrix} e^\theta & 0 \\ 0 & e^{-\theta} \end{bmatrix}$$

since Z has eigenvectors $|0\rangle$ and $|1\rangle$.

Example A.1.36. 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:

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \dots = \sum_{i=0}^{\infty} \frac{A^n}{n!}$$

Note that A must be a square matrix in order for the matrix multiplications and additions to be defined.

Note that matrix exponential is defined for all the kinds of matrices and not necessarily for Normal matrices or non-defective matrices (matrices with the full set

of eigenvectors). A matrix to the 0th power is Identity matrix. Note that as we go for larger and larger powers of matrices it goes to null matrix. As we take higher and higher matrix powers we approach a stable value of a matrix (i.e. all the entries in the matrix reach a stable value).

Example A.1.37. Consider the matrix A which does not have n independent eigen vectors.

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

Clearly, its eigen values are $\lambda = 0, 0$ since its a diagonal matrix and it has only one eigen vector corresponding to $\lambda = 0$ which is $|v_1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Now in order to say find the exponential of this matrix e^A , we use the definition of the exponential series as

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \dots = \sum_{i=0}^{\infty} \frac{A^n}{n!}$$

Now calculating A^2 we get,

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

Thus, the series reduces to

$$e^A = I + A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

It is in general very difficult to calculate the matrix exponential (or matrix function), but in the case when we restrict ourselves to operator A which is a non-defective matrix (i.e. has full independent set of eigenvectors), thus, $A = V\Lambda V^{-1}$, it becomes

easy to compute the exponential of a matrix.

$$\begin{aligned}
 e^A &= I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \dots = \sum_{i=0}^{\infty} \frac{A^n}{n!} \\
 e^A &= I + V\Lambda V^{-1} + \frac{(V\Lambda V^{-1})^2}{2!} + \frac{(V\Lambda V^{-1})^3}{3!} + \dots = \sum_{i=0}^{\infty} \frac{(V\Lambda V^{-1})^n}{n!} \\
 &= V \left(I + \Lambda + \frac{\Lambda^2}{2!} + \frac{(\Lambda)^3}{3!} + \dots \right) V^{-1} = V \left(\sum_{n=0}^{\infty} \frac{\Lambda^n}{n!} \right) V^{-1} \\
 &= Ve^{\Lambda}V^{-1} \\
 &= V \begin{pmatrix} e^{\lambda_1} & 0 & \dots & 0 \\ 0 & e^{\lambda_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & e^{\lambda_n} \end{pmatrix} V^{-1}
 \end{aligned}$$

Note that in the case of A being a normal matrix (or Hermitian matrix) $A = V\Lambda V^T$. In other words, 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:

$$e^A = \begin{pmatrix} \sum_{i=1}^{\infty} \frac{\lambda_1^n}{n!} & 0 & \dots & 0 \\ 0 & \sum_{i=1}^{\infty} \frac{\lambda_2^n}{n!} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sum_{i=1}^{\infty} \frac{\lambda_n^n}{n!} \end{pmatrix}$$

$$e^A = \begin{pmatrix} e^{\lambda_1} & 0 & \dots & 0 \\ 0 & e^{\lambda_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & e^{\lambda_n} \end{pmatrix}$$

Example A.1.38. The square root and logarithm of the matrix

$$X = \begin{bmatrix} 4 & 3 \\ 3 & 4 \end{bmatrix} = 7 \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} + \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$$

Its square root can be given by

$$\begin{aligned} \sqrt{X} &= \sqrt{7} \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} + \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \\ \sqrt{X} &= \begin{bmatrix} \frac{\sqrt{7}+1}{2} & \frac{\sqrt{7}-1}{2} \\ \frac{\sqrt{7}-1}{2} & \frac{\sqrt{7}+1}{2} \end{bmatrix} \end{aligned}$$

and similarly logarithm is:

$$\log(X) = \begin{bmatrix} \frac{\log(7)}{2} & \frac{\log(7)}{2} \\ \frac{\log(7)}{2} & \frac{\log(7)}{2} \end{bmatrix}$$

Some results:

- $e^{A^\dagger} = (e^A)^\dagger$

Proof:

$$\begin{aligned} e^A &= \sum_{k=0}^{\infty} \frac{A^k}{k!} \\ \implies (e^A)^\dagger &= \sum_{k=0}^{\infty} \frac{1}{k!} (A^k)^\dagger \\ &= \sum_{k=0}^{\infty} \frac{(A^\dagger)^k}{k!} \\ &= e^{A^\dagger} \\ \implies (e^A)^\dagger &= e^{A^\dagger} \end{aligned}$$

- $e^{A+B} = e^A e^B$ if $[A, B] = 0$

Proof:

$$\begin{aligned}
 e^A e^B &= \left(\sum_{k=0}^{\infty} \frac{A^k}{k!} \right) \left(\sum_{l=0}^{\infty} \frac{B^l}{l!} \right) \\
 &= \sum_{k=0}^{\infty} \sum_{l=0}^{\infty} \frac{A^k B^l}{k! l!} \\
 &= \sum_{m=0}^{\infty} \sum_{l=0}^m \frac{A^m B^{m-l}}{m! (m-l)!} \\
 &= \sum_{m=0}^{\infty} \frac{1}{m!} \sum_{l=0}^m \frac{m!}{m! (m-l)!} A^m B^{m-l} \\
 &= \sum_{m=0}^{\infty} \frac{(A+B)^m}{m!} \\
 &= e^{A+B}
 \end{aligned}$$

If $[A, B] = 0$ i.e. A and B commute.

- $e^{\iota A}$ is unitary if A is Hermitian and $\det(e^{\iota A}) = e^{\iota \text{Tr}(H)}$.

Proof:

Since A is hermitian ($A = A^\dagger$). Now to prove that $e^{\iota A}$ is unitary we need to show that $(e^{\iota A})^\dagger (e^{\iota A}) = I$. Now, using the property that $(e^A)^\dagger = e^{A^\dagger} \implies (e^{\iota A})^\dagger = e^{-\iota A^\dagger} = e^{-\iota A}$, since A is Hermitian. Note that $(-\iota A)(\iota A) = (\iota A)(-\iota A) = A^2$, thus they commute. Hence, using the property that $e^A e^B = e^{A+B}$ if $[A, B] = 0$ i.e. A and B commute, we get,

$$e^{-\iota A} e^{\iota A} = e^{-\iota A + \iota A} = e^0 = I$$

Thus, $e^{\iota A}$ is Unitary if A is Hermitian.

- $(e^A)^{-1} = e^{-A}$

- $\frac{d}{dt} e^{At} y(0) = A e^{At} y(0) = e^{At} A y(0)$

Proof:

$$\begin{aligned}
e^{At}y(0) &= \left(I + At + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \dots \right) y(0) = \sum_{i=0}^{\infty} \frac{(At)^n}{n!} y(0) \\
\frac{d}{dt} e^{At}y(0) &= \left(A + A^2t + \frac{A^3t^2}{2!} + \dots \right) y(0) \\
&= A(I + At + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \dots) y(0) \\
&= Ae^{At}y(0)
\end{aligned}$$

- $\int_0^t e^{At} dt = A^{-1}[e^{At} - I] = [e^{At} - I]A^{-1}$

A.1.14 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

- **Cyclic Property:** $Tr(AB) = Tr(BA)$.

Let matrix $A = [a_{ij}]_{m \times n}$ and $B = [b_{ij}]_{n \times m}$, Then,

$$tr(AB) = \sum_{i=1}^m \sum_{j=1}^n a_{ij}b_{ji}$$

Similarly,

$$tr(BA) = \sum_{i=1}^n \sum_{j=1}^m b_{ij}a_{ji}$$

Now,

$$\begin{aligned}
 \text{tr}(BA) &= \sum_{i=1}^n \sum_{j=1}^m b_{ij} a_{ji} \\
 &= \sum_{i=1}^n \sum_{j=1}^m a_{ji} b_{ij} \\
 &= \sum_{j=1}^m \sum_{i=1}^n a_{ji} b_{ij} \\
 &= \sum_{i=1}^m \sum_{j=1}^n a_{ij} b_{ji} \\
 \text{tr}(BA) &= \text{tr}(AB)
 \end{aligned}$$

Hence, proved that $\text{tr}(AB) = \text{tr}(BA)$.

- $\text{Tr}(A + B) = \text{Tr}(A) + \text{Tr}(B)$ and if $\text{Tr}(zA) = z\text{Tr}(A)$ for some $z \in \mathbb{C}$.

$$\text{Tr}(A + B) \sum_i a_{ii} + b_{ii} = \sum_i a_{ii} + \sum_i b_{ii} = \text{Tr}(A) + \text{Tr}(B)$$

and

$$\text{Tr}(zA) = \sum_i za_{ii} = z \sum_i a_{ii} = z\text{Tr}(A)$$

- $\text{Tr}(A^\dagger) = \text{Tr}(A)$
- If $\text{Tr}(AA^\dagger) = 0$ then $A = 0$ (Null matrix)
- $\|v\|^2 = \text{Tr}(|v\rangle \langle v|) = \langle v|v \rangle$
- $\text{Tr}(ABC) = \text{Tr}(BCA) = \text{Tr}(CAB)$
- From the cyclic property it follows that the trace of matrix is invariant under the similarity transformation, $A \rightarrow UAU^\dagger$, as $\text{Tr}(UAU^\dagger) = \text{Tr}((U^\dagger U)A) = \text{Tr}(A)$. Thus, trace of an operator is the trace of any matrix representation of A . The invariance of the trace under unitary similarity transformations ensures that the trace of an operator is well defined.

- Suppose $|\psi\rangle$ is a unit vector and A is an arbitrary operator. To evaluate $\text{Tr}(A|\psi\rangle\langle\psi|)$ Using the Gram-Schmidt Procedure to extend $|\psi\rangle$ to an orthonormal basis $|i\rangle$ which includes $|\psi\rangle$ as the first element. Then we have

$$\begin{aligned}\text{Tr}(A|\psi\rangle\langle\psi|) &= \sum_i \langle i|A|\psi\rangle\langle\psi|i\rangle \\ &= \langle\psi|A\psi\rangle\end{aligned}$$

A.1.15 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 α .
- $\|x + y\| \leq \|x\| + \|y\|$ (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 α_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 ∞ -norm of a vector is defined as:

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$$

The ∞ -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.1.16 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, $\|A\|_1 =$ maximum absolute column sum of the matrix A.
- For any matrix A, $\|A\|_\infty =$ maximum absolute row sum of the matrix A.
- For any matrix A, $\|A\|_2 =$ 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.1.17 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 σ_{max} and σ_{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 λ_{max} and λ_{min} are in the absolute sense.

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 define a linear operator $A \otimes B$ on $V \otimes W$ by the equation

$$(A \otimes B)(|v\rangle \otimes |w\rangle) = A|v\rangle \otimes B|w\rangle$$

Consider the tensor products $X = A \otimes B$ and $Y = C \otimes D$. We have:

$$X = A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix},$$

$$Y = C \otimes D = \begin{pmatrix} c_{11}D & c_{12}D & \cdots & c_{1r}D \\ c_{21}D & c_{22}D & \cdots & c_{2r}D \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1}D & c_{n2}D & \cdots & c_{nr}D \end{pmatrix}.$$

To find $(A \otimes B)(C \otimes D)$, we multiply these matrices.

Expanding this product:

$$(A \otimes B)(C \otimes D) = \begin{bmatrix} \sum_{i=1}^n a_{1i}c_{i1}BD & \sum_{i=1}^n a_{1i}c_{i2}BD & \cdots & \sum_{i=1}^n a_{1i}c_{ir}BD \\ \sum_{i=1}^n a_{2i}c_{i1}BD & \sum_{i=1}^n a_{2i}c_{i2}BD & \cdots & \sum_{i=1}^n a_{2i}c_{ir}BD \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n a_{ni}c_{i1}BD & \sum_{i=1}^n a_{ni}c_{i2}BD & \cdots & \sum_{i=1}^n a_{ni}c_{ir}BD \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^n a_{1i}c_{i1} & \sum_{i=1}^n a_{1i}c_{i2} & \cdots & \sum_{i=1}^n a_{1i}c_{ir} \\ \sum_{i=1}^n a_{2i}c_{i1} & \sum_{i=1}^n a_{2i}c_{i2} & \cdots & \sum_{i=1}^n a_{2i}c_{ir} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n a_{ni}c_{i1} & \sum_{i=1}^n a_{ni}c_{i2} & \cdots & \sum_{i=1}^n a_{ni}c_{ir} \end{bmatrix} \otimes BD$$

$$= AC \otimes BD$$

This simplifies to:

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

Thus, we have shown that:

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

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 extended 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.

For say four matrices A, B, C and D , we have

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

given that AC and BD are defined.

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 ψ 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.

Proof: Let A and B be two unitary matrices ($AA^\dagger = I$ and $BB^\dagger = I$). Thus,

$$((A \otimes B)^\dagger)(A \otimes B) = (A^\dagger \otimes B^\dagger)(A \otimes B) = (AA^\dagger \otimes BB^\dagger) = (I \otimes I) = I$$

Similarly, the other two properties given below can be proven.

- 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:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} \otimes \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \\ \beta \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \alpha^2 \\ \alpha\beta \\ \alpha\beta \\ \beta^2 \end{pmatrix}$$

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:

$$\begin{pmatrix} \iota \\ 0 \\ 3 \\ 7 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \\ 3 \\ 7 \end{pmatrix} = \begin{pmatrix} \iota \begin{pmatrix} 1 \\ 0 \\ 3 \\ 7 \end{pmatrix} \\ 7 \begin{pmatrix} 1 \\ 0 \\ 3 \\ 7 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \iota \\ 0 \\ 3\iota \\ 7\iota \\ 7 \\ 0 \\ 21 \\ 49 \end{pmatrix}$$

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:

$$A = |00\rangle\langle 1| + 3|01\rangle\langle 0| + \iota|01\rangle\langle 1| + 7|10\rangle\langle 0| + 13|11\rangle\langle 1|$$

Example A.2.7. Let $|\psi\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$. Write out $|\psi\rangle^{\otimes 2}$ and $|\psi\rangle^{\otimes 3}$ explicitly, both in terms of tensor products like $|0\rangle|1\rangle$, and using the Kronecker product.

Using the definition of tensor product, we have:

$$|\psi\rangle^{\otimes 2} = \frac{|0\rangle|0\rangle + |0\rangle|1\rangle + |1\rangle|0\rangle + |1\rangle|1\rangle}{2} = \begin{bmatrix} 1/\sqrt{2} & [1/\sqrt{2}] \\ 1/\sqrt{2} & [1/\sqrt{2}] \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$$

Similarly,

$$\begin{aligned} |\psi\rangle^{\otimes 3} &= \frac{|0\rangle|0\rangle|0\rangle + |0\rangle|1\rangle|0\rangle + |0\rangle|1\rangle|1\rangle + |1\rangle|0\rangle|0\rangle + |1\rangle|0\rangle|1\rangle + |1\rangle|1\rangle|0\rangle + |1\rangle|1\rangle|1\rangle}{2\sqrt{2}} \\ &= |\psi\rangle \otimes |\psi\rangle^{\otimes 2} = \begin{bmatrix} \frac{1}{\sqrt{2}} & [1/2] \\ \frac{1}{\sqrt{2}} & [1/2] \\ \vdots & \vdots \\ \frac{1}{\sqrt{2}} & [1/2] \end{bmatrix} = \begin{bmatrix} \frac{1}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} \\ \vdots \\ \frac{1}{2\sqrt{2}} \end{bmatrix} \end{aligned}$$

Example A.2.8. Show that the transpose, complex conjugation and adjoint operations distribute over the tensor product.

$$(A \otimes B)^* = A^* \otimes B^*; (A \otimes B)^T = A^T \otimes B^T; (A \otimes B)^\dagger = A^\dagger \otimes B^\dagger$$

Using the Kronecker product of $A \otimes B$, we have:

$$\begin{aligned} (A \otimes B)^* &= \begin{bmatrix} A_{11}B & A_{12}B & \dots & A_{1n}B \\ A_{21}B & A_{22}B & \dots & A_{2n}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{m1}B & A_{m2}B & \dots & A_{mn}B \end{bmatrix}^* = \begin{bmatrix} A_{11}^*B^* & A_{12}^*B^* & \dots & A_{1n}^*B^* \\ A_{21}^*B^* & A_{22}^*B^* & \dots & A_{2n}^*B^* \\ \vdots & \vdots & \vdots & \vdots \\ A_{m1}^*B^* & A_{m2}^*B^* & \dots & A_{mn}^*B^* \end{bmatrix} = A^* \otimes B^* \\ (A \otimes B)^T &= \begin{bmatrix} A_{11}B & A_{12}B & \dots & A_{1n}B \\ A_{21}B & A_{22}B & \dots & A_{2n}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{m1}B & A_{m2}B & \dots & A_{mn}B \end{bmatrix}^T = \begin{bmatrix} A_{11}B^T & A_{21}B^T & \dots & A_{n1}B^T \\ A_{12}B^T & A_{22}B^T & \dots & A_{n2}B^T \\ \vdots & \vdots & \vdots & \vdots \\ A_{1m}B^T & A_{2m}B^T & \dots & A_{nm}B^T \end{bmatrix} = A^T \otimes B^T \end{aligned}$$

The relation for the distributivity of the hermitian conjugate over the tensor product then follows from the former two relations:

$$(A \otimes B)^\dagger = ((A \otimes B)^T)^* = (A^T \otimes B^T)^* = (A^T)^* \otimes (B^T)^* = A^\dagger \otimes B^\dagger$$

Example A.2.9. Show that the tensor product of two unitary operators is unitary.

Suppose A, B are unitary. Then, $A^\dagger A = I$ and $B^\dagger B = I$.

$$(A \otimes B)^\dagger (A \otimes B) = (A^\dagger \otimes B^\dagger) = (A^\dagger A \otimes B^\dagger B) = I \otimes I$$

Example A.2.10. Show that the tensor product of two Hermitian operators is Hermitian.

Suppose A, B are Hermitian. Then, $A^\dagger = A$ and $B^\dagger = B$. Then, using the previous results, we have:

$$(A \otimes B)^\dagger = A^\dagger \otimes B^\dagger = A \otimes B$$

Example A.2.11. Show that the tensor product of two positive operators is positive.

Suppose A and B are positive operators. We then have that $\langle v|A|v\rangle \geq 0$ and $\langle w|B|w\rangle \geq 0$. Therefore, for any $|v\rangle \otimes |w\rangle$:

$$(\langle v| \otimes \langle w|)(A \otimes B)(|v\rangle \otimes |w\rangle) = (\langle v|A|v\rangle \otimes \langle w|B|w\rangle) = \langle v|A|v\rangle \langle w|B|w\rangle \geq 0$$

Example A.2.12. Show that the tensor product of two projectors is a projector.

Let P_1, P_2 be projectors. We then have that $P_1^2 = P_1$ and $P_2^2 = P_2$. Therefore,

$$(P_1 \otimes P_2)^2 = (P_1 \otimes P_2)(P_1 \otimes P_2) = P_1^2 \otimes P_2^2 = P_1 \otimes P_2$$

so $P_1 \otimes P_2$ is a projector.

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,

$$ac = \frac{1}{2}, ad = 0, bc = -\frac{1}{2}, bd = \frac{1}{\sqrt{2}}$$

Thus, $a = \frac{1}{\sqrt{2}}, b = 0, c = 1, d = \frac{1}{\sqrt{2}}$. Thus, the tensor product can be factorized as $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle|1\rangle + \frac{1}{\sqrt{2}}|1\rangle|0\rangle$.

Consider a $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle$, then we can factorize the tensor product by 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:

$$|\psi\rangle = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle$$

Thus,

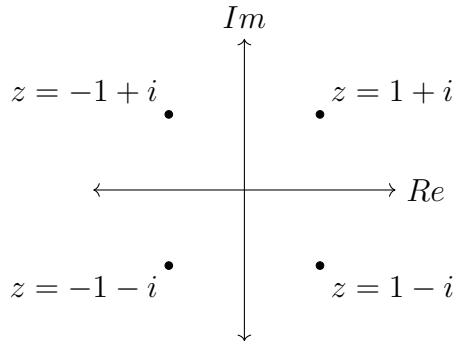
$$ac = \frac{1}{\sqrt{2}}, ad = 0, bc = 0, bd = \frac{1}{\sqrt{2}}$$

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 $0 = 0 + 0\imath$
- 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:

$$z_1z_2 = (a_1 + \imath b_1)(a_2 + \imath b_2) = a_1a_2 - a_1b_2 + \imath(a_1b_2 + a_2b_1)$$

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: $\alpha(z_1 z_2) = \alpha z_1 z_2$
- 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 $\iota\theta$ in place of x , we get:

$$e^{\iota\theta} = 1 + \iota\theta + \frac{(\iota\theta)^2}{2!} + \frac{(\iota\theta)^3}{3!} + \dots = \sum_{n=0}^{\infty} \frac{(\iota\theta)^n}{n!}$$

$$e^{\iota\theta} = 1 + \iota\theta - \frac{\theta^2}{2!} - \iota\frac{\theta^3}{3!} + \dots = \cos(\theta) + \iota \sin(\theta)$$

On rearranging the terms we get:

$$e^{\iota\theta} = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \dots + \iota(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \dots)$$

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 $e^{\iota\theta} = \cos(\theta) + \iota \sin(\theta)$. Hence, on substituting this in the polar form of a complex number we get:

$$z = a + \iota b = \sqrt{a^2 + b^2} \left(\cos \left(\tan^{-1} \frac{b}{a} \right) + \iota \sin \left(\tan^{-1} \frac{b}{a} \right) \right) = r(\cos(\theta) + \iota \sin(\theta)) = r e^{\iota\theta}$$

where $r = |z|$ and $\theta = \tan^{-1} \left(\frac{b}{a} \right)$. Using, this we can also write:

$$\cos \theta = \frac{e^{\iota\theta} + e^{-\iota\theta}}{2}$$

$$\sin \theta = \frac{e^{\iota\theta} - e^{-\iota\theta}}{2\iota}$$

A.4 Probability Theory

A.5 Calculus

Exponential of matrices generally arise in Quantum mechanics in the form of solution to Schrodinger equation when the matrix H is constant. Schrodinger equation is given as

$$\imath\hbar \frac{\partial}{\partial t} |\psi\rangle = \hat{H} |\psi\rangle$$

with the initial condition at $t = 0$ as $|\psi(0)\rangle$. In the case, when the Hamiltonian \hat{H} is a constant matrix, the above equation can be simplified to

$$\begin{aligned}\frac{\partial}{\partial t} |\psi(t)\rangle &= -\frac{\imath\hat{H}}{\hbar} |\psi\rangle \\ |\psi(t)\rangle &= \int_0^t -\frac{\imath\hat{H}}{\hbar} |\psi\rangle dt \\ |\psi(t)\rangle &= e^{-\imath\hat{H}t/\hbar} |\psi(0)\rangle\end{aligned}$$

A.6 Group Theory

The mathematical theory of groups is useful at several points in the study of quantum computation and quantum information. the generalization of the order-finding, factoring, and period finding algorithms is based on the hidden subgroup problem; the stabilizer formalism for quantum error -correction is based on some elementary group theory. The number theory uses the properties of the group Z_n^* . And, implicitly, the quantum circuits used throughout the book are an example of the use of Lie groups. In this appendix we review some basic material on group theory. We summarize many of the fundamental concepts and provide important definitions, but do not attempt to explain very much, as group theory is a vast subject!

A.6.1 Basic Definitions

Definition A.6.1. Group: A group (G, \cdot) is a non-empty set G with a binary group multiplication operation ‘.’, with the following properties:

- (**Closure**): $g_1 \cdot g_2 \in G$ for all $g_1, g_2 \in G$

- (**Associativity**) $(g_1 \cdot g_2) \cdot g_3 = g_1 \cdot (g_2 \cdot g_3)$, for all $g_1, g_2, g_3 \in G$
- (**Identity**) There exists $e \in G$ such that $\forall g \in G, g \cdot e = e \cdot g = g$
- (**Inverses**) $\forall g \in G$, there exists $g^{-1} \in G$ such that $g \cdot g^{-1} = e$ and $g^{-1} \cdot g = e$.

Note we often leave out \cdot and write $g_1 \cdot g_2$ as simply $g_1 g_2$. We also often refer to the group G without referring explicitly to its multiplication operation, but it must be defined.

A group G is *finite* if the number of elements in G is finite. The order of a finite group G is the number of elements it contains, denoted as $|G|$. A group G is said to be abelian if $g_1 g_2 = g_2 g_1$ for all $g_1, g_2 \in G$. A simple example of a finite Abelian group is the additive group \mathbb{Z}_n of integers modulo n , with ‘multiplication’ operation the operation of modular addition. It is easily checked that this operation satisfies the closure and associativity axioms; there is an identity element, 0, since $x + 0 = x \pmod{n}$ for all x , and every $x \in G$ has an inverse, $n - x$ since $x + (n - x) = 0 \pmod{n}$.

The order of an element $g \in G$ is the smallest positive integer r such that g^r (a multiple with itself r times) equals the identity element e .

A subgroup H of G is a subset of G which forms a group under the same group multiplication operation as G .

Theorem A.6.1. (Lagrange’s Theorem) *If H is a subgroup of a finite group G then $|H|$ divides $|G|$.*

Proof.

□

Example A.6.1. Prove that for any element g of a finite group, there always exists a positive integer r such that $g^r = e$. That is, every element of such a group has an order.

Example A.6.2. Show that the order of an element $g \in G$ divides $|G|$.

If g_1 and g_2 are elements of G , then the conjugate of g_2 with respect to g_1 is the element $g_1^{-1}g_2g_1$. If H is a subgroup of G , then it is known as normal group if $g^{-1}Hg = H$ for all $g \in G$. The conjugacy class G_x of an element x in a group G is defined by $G_x \equiv \{g^{-1}xg | g \in G\}$.

Example A.6.3. Show that if $y \in G_x$ then $G_y = G_x$.

Example A.6.4. Show that if x is an element of an Abelian group G then $G_x = \{x\}$.

Since $x \in G$ is an Abelian group, it follows the property that $g_1g_2 = g_2g_1$ for all $g_1, g_2 \in G$. Thus, $G_x = \{g^{-1}xg | g \in G\}$ which implies that $G_x = \{g^{-1}gx | g \in G\} = \{x | g \in G\}$ since $g^{-1}g = e$. Thus, $G_x = \{x\}$.

An interesting example of a group which is not Abelian is the Pauli group on n qubits. For a single qubit, the pauli group is defined to consist of all the Pauli matrices, with multiplicative factors $\pm 1, \pm i$ allowed by the definition:

$$G_1 \equiv \{\pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\}$$

This set of matrices forms a group under the operation of matrix multiplication. You might wonder why we don't omit the multiplicative factors ± 1 and $\pm i$; the reason these are included is to ensure that G_1 is closed under multiplication, and thus forms a legitimate group. The general Pauli group on n qubits is defined to consist of all n -fold tensor products of Pauli matrices, and again we allow multiplicative factors of $\pm 1, \pm i$.

A.6.2 Generators

The study of groups is often greatly simplified by the use of a set of group generators for the group being studied. A set of elements g_1, \dots, g_l in a group G is said to generate the group G if every element of G can be written as a product of (possibly repeated) elements from the list g_1, \dots, g_l , and we write $G = \langle g_1, \dots, g_l \rangle$. For example, $G_1 = \langle X, Z, iI \rangle$, since every element in G can be written as a product of X, Z and iI . On the other hand, $\langle X \rangle = \{I, X\}$, a subgroup of G_1 , since not every element of G_1 can be written as a power of X . The notation $\langle \dots \rangle$ which we use for group generators and may potentially be confused with the notation for observable averages; however, in practice it is always clear context how the notation is being used,

The great advantage of using generators to describe groups is that they provide a compact means of describing the group. Suppose G has size $|G|$. Then it is pretty easy to show that there is a set of $\log(|G|)$ generators generating G . To see this, suppose g_1, \dots, g_l is a set of elements in a group G , and g is not an element of $\langle g_1, \dots, g_l \rangle$. Let $f \in \langle g_1, \dots, g_l \rangle$. Then $fg \notin \langle g_1, \dots, g_l \rangle$, since if it were then we would have $g = f^{-1}fg \in \langle g_1, \dots, g_l \rangle$, which we would know is false by assumption. Thus for each element $f \in \langle g_1, \dots, g_l \rangle$ there is an element fg which is in $\langle g_1, \dots, g_l, g \rangle$ but not in $\langle g_1, \dots, g_l \rangle$. Thus adding the generator g to $\langle g_1, \dots, g_l \rangle$ doubles (or more) the size of group being generated, from which we conclude that G must have a set of generators containing at most $\log(|G|)$ elements.

A.7 Number Theory

The following section will be very useful in understanding the concepts behind Order finding algorithm and will form the foundation for the Shor's algorithm.

A.7.1 Fundamentals

We will start off by agreeing about a few conventions for nomenclature and notation. The set of integers is the set $\{\dots, -2, -1, 0, 1, 2, \dots\}$, denoted \mathbf{Z} . We may occasionally refer to the natural numbers, meaning non-negative integer, but more often we'll say non-negative integer or positive integer, in order to make distinction between the case when zero is included, and when zero is not included.

Suppose n is an integer. An integer d divides n (written as $d|n$) if there exists an integer k such that $n = dk$. We say in this case that d is a factor or divisor of n . Notice that 1 and n are always factors of n . When d does not divide (is not a factor of) n we write $d \nmid n$. For example, $3|6$ and $3|18$, but $3 \nmid 5$ and $3 \nmid 7$.

Example A.7.1. (Transitivity) If $a|b$ and $b|c$ then $a|c$. As $a|b \implies b = pa$ and $b|c \implies c = qb$ for some integers q and p . Thus, substituting the value of b gives $c = qb = (qp)a \implies a|c$ for some integer $pq = p \times q$ (since multiplication of two integers is an integer).

Example A.7.2. Show that if $d|a$ and $d|b$ then d also divides linear combination of a and b , $ax + by$, where x and y are integers.

$d|a \implies a = pd$ and $d|b \implies b = qd$ for some integers p and q . Thus, $ax + by = pdx + qdy = (px + qy)d \implies d|(ax + by)$, since $px + qy$ is also an integer where p, q, x, y are all integers.

Example A.7.3. Suppose a and b are positive integers. Show that if $a|b$ then $a \leq b$. Conclude that if $a|b$ and $b|a$ then $a = b$.

Since $a|b$ and a, b are both positive integers thus, $b = pa$ for some positive integer p . Thus, $p \geq 1$, thus $pa \geq a \implies b \geq a$ (since p, a, b are all positive integers. Hence, proved.

If $a|b$ and $b|a$ then $b = pa$ and $a = qb$ for some positive integers p, q . Upon substituting, we get, $b = pqb \implies pq = 1$. Now since both p and q are positive integers such that $p \geq 1$ and $q \geq 1$ and $pq = 1$. Thus, the only solution satisfying this equation is $p = q = 1$. Hence, $a = b$.

Definition A.7.1. A prime number is an integer greater than 1 which has only itself and 1 as factors. The first few prime numbers are 2, 3, 5, 7, 11, 13, 17,

The most important single fact about positive integers is that they may be represented uniquely as a product of factors which are prime numbers. This result is called the fundamental theorem of arithmetic.

Theorem A.7.1. (*Fundamental Theorem of Arithmetic*) *Let a be any integer greater than 1. Then a has a prime factorization of the form*

$$a = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}$$

where p_1, p_2, \dots, p_n are distinct prime numbers, and a_1, \dots, a_n are positive integers. Moreover, this prime factorization is unique, up to the order of the factors.

Proof. Either n is a prime or it is composite; in the former case, there is nothing more to prove. If n is composite, then there exists an integer d satisfying $d|n$ and $1, d < n$. Among all such integers d , choose p_1 to be the smallest (this is possible by the Well-Ordering principle). Then p_1 must be a prime number. Otherwise it too would have a divisor q with $1 < q < p_1$; but then $q|p_1$ and $p_1|n$ imply that $q|n$, which contradicts the choice of p_1 as the smallest divisor, not equal to 1, of n .

We therefore may write $n = p_1 n_1$ where p_1 is prime and $1 < n_1 < n$. If n_1 happens to be a prime, then we have our representation. In the contrary case, the argument is repeated to produce a second prime p_2 such that $n_1 = p_2 n_2$: that is,

$$n = p_1 p_2 n_2 \quad 1 < n_2 < n_1$$

If n_2 is prime, then it is not necessary to go further. Otherwise, write $n_2 = p_3 n_3$, with p_3 is a prime:

$$n = p_1 p_2 p_3 n_3 \quad 1 < n_3 < n_2$$

The decreasing sequence

$$n > n_1 > n_2 > \dots > 1$$

cannot continue indefinitely, so that after a finite number of steps n_{k-1} is a prime, call it, p_k . This leads to the prime factorization

$$n = p_1 p_2 \cdots p_k$$

To establish the second part of the proof - the uniqueness of the prime factorization - let us suppose that the integer n can be represented as a product of primes in two ways: say

$$n = p_1 p_2 \cdots p_r = q_1 q_2 \cdots q_s \quad r \leq s$$

where the p_i and q_i are all primes, written in increasing magnitude so that

$$p_1 \leq p_2 \leq \cdots \leq p_r \quad q_1 \leq q_2 \leq \cdots \leq q_s$$

Because $p_1|q_1q_2\dots q_s$, tells us that $p_1 = q - k$ for some k : but then $p_1 \geq q_1$. Similar reasoning gives $q_1 \geq p_1$, whence $p_1 = q_1$. We may cancel this common factor and obtain

$$p_2p_3\dots p_r = q_2q_3\dots q_s$$

Now repeat the process to get $p_2 = q_2$ and, in turn

$$p_3p_4\dots p_r = q_3q_4\dots q_s$$

Continue in this fashion. If the inequality $r < s$ were to hold, we would eventually arrive at

$$1 = q_{r+1}q_{r+2}\dots q_s$$

which is absurd, because each $q_j > 1$. Hence, $r = s$ and

$$p_1 = q_1 \quad p_2 = q_2, \dots, p_r = q_r$$

making the two factorizations of n identical. □

Example A.7.4. Find the prime factorization of 697 and 36300.

Prime factorization of $697 = 17 \times 41$ and $36300 = 2^2 \times 3 \times 5^2 \times 11^2$.

A.7.2 Modular Arithmetic and Euclid's Algorithm

We're all thoroughly familiar with the techniques of ordinary arithmetic. Another type of arithmetic, modular arithmetic, is extremely useful in understanding the properties of numbers. We assume that you are familiar with the elementary ideas of modular arithmetic, and so will quickly breeze through the basic ideas and notation, before coming to more advanced theory.

Modular arithmetic can be thought of as the arithmetic of remainders. If we divide 18 by 7 we get the answer 2, with a remainder 4. More formally, given any positive integers x and n , x can be written (uniquely) in the form

$$x = kn + r$$

where k is a non-negative integer, the result of dividing x by n , and the remainder r lies in the range 0 to $n - 1$, inclusive. Modular arithmetic is simply ordinary arithmetic in which we only pay attention to remainders. We use the notation $(\text{mod } n)$ to indicate that we are working in modular arithmetic. For instance $2 = 5 = 8 = 11 \pmod{3}$, because 2, 5, 8 and 11 all have the same remainder 2 when divided by 3. The appellation $(\text{mod } n)$ reminds us that we are working in modular arithmetic,

with respect to the number n . The congruence relation satisfies all the conditions of an equivalence relation:

Reflexivity: $a \equiv a \pmod{m}$

Symmetry: $a \equiv b \pmod{m}$ if $b \equiv a \pmod{m}$.

Transitivity: If $a \equiv b \pmod{m}$ and $b \equiv c \pmod{m}$, then $a \equiv c \pmod{m}$.

If $a_1 \equiv b_1 \pmod{m}$ and $a_2 \equiv b_2 \pmod{m}$, or if $a \equiv b \pmod{m}$, then:

- $a + k \equiv b + k \pmod{m}$ for any integer k (compatibility with translation),
- $ka \equiv kb \pmod{m}$ for any integer k (compatibility with scaling),
- $ka \equiv kb \pmod{km}$ for any integer k ,
- $a_1 + a_2 \equiv b_1 + b_2 \pmod{m}$ (compatibility with addition),
- $a_1 - a_2 \equiv b_1 - b_2 \pmod{m}$ (compatibility with subtraction),
- $a_1 a_2 \equiv b_1 b_2 \pmod{m}$ (compatibility with multiplication),
- $a^k \equiv b^k \pmod{m}$ for any non-negative integer k (compatibility with exponentiation),
- $p(a) \equiv p(b) \pmod{m}$, for any polynomial $p(x)$ with integer coefficients (compatibility with polynomial evaluation).

If $a \equiv b \pmod{m}$, then it is generally false that $k^a \equiv k^b \pmod{m}$. However, the following is true:

- If $c \equiv d \pmod{\varphi(m)}$, where φ is Euler's totient function, then $a^c \equiv a^d \pmod{m}$ —provided that a is coprime with m .

For cancellation of common terms, we have the following rules:

- If $a + k \equiv b + k \pmod{m}$, where k is any integer, then $a \equiv b \pmod{m}$.
- If $ka \equiv kb \pmod{m}$ and k is coprime with m , then $a \equiv b \pmod{m}$.
- If $ka \equiv kb \pmod{km}$ and $k \neq 0$, then $a \equiv b \pmod{m}$.

The last rule can be used to move modular arithmetic into division. If b divides a , then

$$\frac{a}{b} \pmod{m} = \frac{a \pmod{bm}}{b}.$$

The modular multiplicative inverse is defined by the following rules:

- **Existence:** There exists an integer denoted a^{-1} such that $aa^{-1} \equiv 1 \pmod{m}$ if and only if a is coprime with m . This integer a^{-1} is called a modular multiplicative inverse of a modulo m .
- If $a \equiv b \pmod{m}$ and a^{-1} exists, then $a^{-1} \equiv b^{-1} \pmod{m}$ (compatibility with multiplicative inverse, and, if $a = b$, uniqueness modulo m).
- If $ax \equiv b \pmod{m}$ and a is coprime to m , then the solution to this linear congruence is given by $x \equiv a^{-1}b \pmod{m}$.

The multiplicative inverse $x \equiv a^{-1} \pmod{m}$ may be efficiently computed by solving Bézout's equation $ax + my = 1$ for x, y , by using the Extended Euclidean algorithm.

In particular, if p is a prime number, then a is coprime with p for every a such that $0 < a < p$; thus, a multiplicative inverse exists for all a that is not congruent to zero modulo p .

A.7.3 Advanced Properties:

Some of the more advanced properties of congruence relations are the following:

Fermat's Little Theorem: If p is prime and does not divide a , then

$$a^{p-1} \equiv 1 \pmod{p}.$$

Euler's Theorem: If a and m are coprime, then

$$a^{\varphi(m)} \equiv 1 \pmod{m},$$

where φ is Euler's totient function.

A simple consequence of Fermat's Little Theorem is that if p is prime, then

$$a^{-1} \equiv a^{p-2} \pmod{p}$$

is the multiplicative inverse of $0 < a < p$. More generally, from Euler's Theorem, if a and m are coprime, then

$$a^{-1} \equiv a^{\varphi(m)-1} \pmod{m}.$$

Hence, if $ax \equiv 1 \pmod{m}$, then

$$x \equiv a^{\varphi(m)-1} \pmod{m}.$$

Another simple consequence is that if $a \equiv b \pmod{\varphi(m)}$, where φ is Euler's totient function, then

$$ka \equiv kb \pmod{m}$$

provided k is coprime with m .

Wilson's Theorem: p is prime if and only if

$$(p-1)! \equiv -1 \pmod{p}.$$

Chinese Remainder Theorem: For any integers a, b and coprime integers m, n , there exists a unique $x \pmod{mn}$ such that

$$x \equiv a \pmod{m} \quad \text{and} \quad x \equiv b \pmod{n}.$$

In fact, the solution is given by

$$x \equiv b \cdot m^{n-1} \pmod{mn} + a \cdot n^{m-1} \pmod{mn},$$

where m^{n-1} is the inverse of m modulo n and n^{m-1} is the inverse of n modulo m .

Lagrange's Theorem: If p is prime and $f(x) = a_0x^d + \dots + a_d$ is a polynomial with integer coefficients such that p is not a divisor of a_0 , then the congruence

$$f(x) \equiv 0 \pmod{p}$$

has at most d non-congruent solutions modulo p .

Primitive Root Modulo m : A number g is a primitive root modulo m if, for every integer a coprime to m , there is an integer k such that

$$g^k \equiv a \pmod{m}.$$

A primitive root modulo m exists if and only if m is equal to 2, 4, p^k or $2p^k$, where p is an odd prime number and k is a positive integer. If a primitive root modulo m exists, then there are exactly $\varphi(\varphi(m))$ such primitive roots, where φ is Euler's totient function.

Quadratic Residue: An integer a is a quadratic residue modulo m if there exists an integer x such that

$$x^2 \equiv a \pmod{m}.$$

Euler's criterion asserts that, if p is an odd prime, and a is not a multiple of p , then a is a quadratic residue modulo p if and only if

$$a^{\frac{p-1}{2}} \equiv 1 \pmod{p}.$$

Addition, multiplication, and subtraction operations for modular arithmetic may all be defined in the obvious way, but it is perhaps not so obvious how to define a division operation. To understand how this may be done we introduce another key concept from number theory, that of the greatest common divisor of two integers. The greatest common divisor of integers a and b is the largest integer which is a divisor of both a and b . We write this number as $\gcd(a, b)$. For example, the greatest common divisor of 18 and 12 is 6. An easy way of seeing this is to enumerate the positive divisors of 18 (1, 2, 3, 6, 9, 18) and 12 (1, 2, 3, 4, 6, 12), and then pick out the largest common element in the two lists. This method is quite inefficient and impractical for large numbers. Fortunately, there is a much more efficient way of working out the greatest common divisor, a method known as Euclid's algorithm.

Theorem A.7.2. (Representation theorem for the gcd) *The greatest common divisor of two integers a and b is the least positive integer that can be written in the form $ax + by$, where x and y are integers.*

Proof. Let $s = ax + by$ be the smallest positive integer that can be written in this form. Since $\gcd(a, b)$ is a divisor of both a and b it is also a divisor of s . It follows that $\gcd(a, b) \leq s$. To complete the proof we demonstrate that $s \leq \gcd(a, b)$ by showing that s is a divisor of both a and b . The proof is by contradiction. Suppose s is not a divisor of a . Then $a = ks + r$, where the remainder r is in the range 1 to $s - 1$. Rearranging this equation and using $s = ax + by$ we see that $r = a(1 - kx) + b(-ky)$ is a positive integer that can be written as a linear combination of a and b , which is smaller than s . But this contradicts the definition of s as the smallest positive integer that can be written as a linear combination of a and b . We conclude that s must divide a . By symmetry s must also be a divisor of b , which completes the proof. \square

Corollary A.7.3. *Suppose c divides both a and b . Then c divides $\gcd(a, b)$.*

Proof. From previous theorem, $\gcd(a, b) = ax + by$ for some integers x and y . Since c divides a and b it must also divide $ax + by$. \square

When does a number, a , have a multiplicative inverse in modular arithmetic? That is, given a and n , when does there exist a b such that $ab = 1 \pmod{n}$? For example, note that $2 \times 3 = 1 \pmod{5}$, so the number 2 has a multiplicative inverse 3 in arithmetic modulo 5. On the other hand, trial and error shows that 2 has no multiplicative inverse modulo 4. Finding multiplicative inverse in modular arithmetic turns out to be related to the gcd by the notion of co-primality: integers a and b are said to be co-prime if their greatest common divisor is 1. For example, 14 and

9 are co-prime, since the positive divisors of 14 are 1, 2, 7 and 14, while 1, 3 and 9 are the positive divisors of 9. The following corollary characterizes the existence of multiplicative inverses in modular arithmetic using co-primality.

Corollary A.7.4. *Let n be an integer greater than 1. An integer a has a multiplicative inverse modulo n if and only if $\gcd(a, n) = 1$, that is, a and n are co-prime.*

Proof. Suppose a has a multiplicative inverse, which we denote a^{-1} , modulo n . Then $aa^{-1} = 1 + kn$ for some integer k , and thus $aa^{-1} + (-k)n = 1$ (since gcd is defined as the smallest positive integer which is 1). From the previous theorem we conclude that $\gcd(a, n) = 1$. Conversely, if $\gcd(a, n) = 1$ then there must exist integers a^{-1} and b such that $aa^{-1} + bn = 1$, and therefore $aa^{-1} \pmod{n}$. \square

Example A.7.5. For p a prime prove that all the integers in the range 1 to $p - 1$ have multiplicative inverse modulo p . Which integers in the range 1 to $p^2 - 1$ do not have multiplicative inverses modulo p^2 ?

From the previous example we see that an integer a has a multiplication inverse modulo n if and only if $\gcd(a, n) = 1$. For a prime p it has factors as 1 and itself. Thus, $\gcd(a, p) = 1$ for all a in the range 1 to $p - 1$ since $p \nmid a$ as $a < p$. Hence, as shown earlier a number a has a multiplication inverse modulo n if and only if $\gcd(a, n) = 1$. Thus, $\forall a$ in the range 1 to $p - 1$ for a prime p have a multiplication inverse modulo p .

The integers which have $\gcd(a, p^2) = 1$ for a in the range 1 to $p^2 - 1$ will have multiplicative inverses modulo p^2 . p^2 has its prime factorization as $p^2 = 1 \times p \times p$. Thus, we are looking for a such that $\gcd(a, p^2) = 1$ for all the a in the range 1 to $p - 1$. Clearly, for $a = p, 2p, 3p, \dots, p(p-1)$, $\gcd(a, p^2) = p$. Thus, except for $a = rp$ for r in the range 1 to $p - 1$, all the others will have $\gcd(a, p^2) = 1$. Thus, $\forall a$ and integer r , $a \neq rp$; $1 \leq r < p$, $\gcd(a, p^2) = 1$ and hence will have a multiplication modulo inverse.

Example A.7.6. Find the multiplicative inverse of 17 modulo 24.

Clearly, $\gcd(17, 24) = 1$, hence a multiplicative inverse exists. The multiplicative inverse is 17 itself. Because $17 \times 17 = 289 = 24 \times 12 + 1$. Thus, the multiplicative inverse of 17 is 17.

This can also be found out using Euclid's algorithm as follows:

$$\begin{aligned}
 24 &= 17 \times 1 + 7 \\
 17 &= 7 \times 2 + 3 \\
 7 &= 3 \times 2 + 1 \\
 3 &= 3 \times 1 \\
 7 - 3 \times 2 &= 1 \\
 -17 \times 2 + 7 \times 5 &= 1 \\
 24 \times 5 + 17 \times (-7) &= 1
 \end{aligned}$$

Thus, $17 \times (-7) = 1 \pmod{24}$, hence -7 is a multiplicative inverse of $17 \implies -7 + 24 = 17$. Thus, $-7 \equiv 17 \pmod{24}$.

Example A.7.7. Find the multiplicative inverse of $n+1$ modulo n^2 , where n is any integer greater than 1.

Since $(n+1)(n-1) \equiv (-1) \pmod{n^2}$ and since $(-1)^2 \equiv 1 \pmod{n^2}$, you have that

$$[(n+1)(n-1)]^2 \equiv 1 \pmod{n^2}$$

Therefore, the inverse of $(n+1)$ is $[(n+1)(n-1)]^2$.

Example A.7.8. (Uniqueness of the inverse) Suppose b and b' are multiplicative inverses of a , modulo n . Prove that $b = b' \pmod{n}$.

Since b and b' are both multiplicative inverses of a , thus, $ab \equiv ab' = 1 \pmod{n}$. Thus, $\gcd(a, n) = 1$

$$\begin{aligned}
 ab &\equiv 1 \pmod{n} \\
 ab' &\equiv 1 \pmod{n} \\
 \implies a(b - b') &\equiv 0 \pmod{n}
 \end{aligned}$$

Now, since $n \nmid a$. Thus, $n|(b - b')$. Hence, $b \equiv b' \pmod{n}$. This is because congruent values are considered as same in modular arithmetic. For example, $3 \times 2 = 6 = 1 \pmod{5}$ and $3 \times 7 = 21 = 1 \pmod{5}$. Thus, what we mean by unique is that $2 = 7 \pmod{5}$. They belong to the same class i.e. same in congruence. thus congruent values are considered as same in modular arithmetic.

The next theorem is the key to Euclid's efficient algorithm for finding the greatest common divisor of two positive integers.

Theorem A.7.5. Let a and b be integers, and let r be the remainder when a is divided by b . Then provided $r \neq 0$.

$$\gcd(a, b) = \gcd(b, r)$$

Proof. We prove the equality by showing that each side divides the other. To prove that the left hand side divides the right note that $r = a - kb$ for some integer k . Since $\gcd(a, b)$ divides a, b and linear combinations of these it follows that $\gcd(a, b)$ divides r . Using the fact that if some integer $c|a$ and $c|b$ then $c|\gcd(a, b)$. Thus, $\gcd(a, b)$ divides $\gcd(b, r)$. To prove that the right hand side divides the left note that $\gcd(b, r)$ divides b and r , and since $a = r + kb$ is a linear combination of b and r it follows that $\gcd(b, r)$ also divides a . Thus, $\gcd(b, r)$ divides $\gcd(a, b)$. \square

Example A.7.9. Explain how to find $\gcd(a, b)$ if the prime factorization of a and b are known. Find the prime factorizations of 6825 and 1430, and use them to compute $\gcd(6825, 1430)$.

Suppose the prime factorization of $a = a_1^{\alpha_1}a_2^{\alpha_2}\dots a_p^{\alpha_p}$ and $b = b_1^{\beta_1}b_2^{\beta_2}\dots b_q^{\beta_q}$. Clearly, all the $a_i^{\alpha_i}$ for $i = 1, \dots, p$ divides a and all the $b_i^{\beta_i}$ for $i = 1, \dots, q$ divides b . We should choose a_i and b_i such that $a_i = b_i$ along with their highest powers possible so that they divide both a and b . Thus, we can get the $\gcd(a, b)$ by this method. Let us now see this through an example by finding $\gcd(6825, 1430)$.

Note that we can write $6825 = 3 \times 5 \times 5 \times 7 \times 13$ and $1430 = 2 \times 5 \times 11 \times 13$. Thus, taking the common prime factors along with their possible highest powers we get their \gcd as $5 \times 13 = 65$. Thus, $\gcd(1430, 6825) = 65$.

Euclid's algorithm for finding the greatest common divisor of positive integers a and b works as follows. first, order a and b so that $a > b$. Divide b into a , with result k_1 and remainder $r_1 : a = k_1b + r_1$. Now using the theorem $\gcd(a, b) = \gcd(b, r_1)$. Next, we perform a second division b playing the role of a , and r_1 playing the role of $b : b = k_2r_1 + r_2$. By Theorem, $\gcd(a, b) = \gcd(b, r_1) = \gcd(r_1, r_2)$. Next, we perform a third divisor with r_1 playing the role of $b : r_1 = k_3r_2 + r_3$. By theorem, $\gcd(a, b) = \gcd(b, r_1) = \gcd(r_1, r_2) = \gcd(r_2, r_3)$. We continue in this manner, each time dividing the most recent remainder by the second most recent remainder, obtaining a new result and remainder. The algorithm by the second most recent remainder, obtaining a new result and remainder. the algorithm halts when we obtain a remainder that is zero, that is, $r_m = k_{m+1}r_{m+1}$ for some m . We have $\gcd(a, b) = \gcd(r_m, r_{m+1}) = r_{m+1}$, so the algorithm returns r_{m+1} .

As an example of the use of Euclid's algorithm we find $\gcd(6825, 1430)$

$$\begin{aligned} 6825 &= 4 \times 1430 + 1105 \\ 1430 &= 1 \times 1105 + 325 \\ 1105 &= 3 \times 325 + 130 \\ 325 &= 2 \times 130 + 65 \\ 130 &= 2 \times 65 \end{aligned}$$

From this we see that $\gcd(6825, 1430) = 65$.

An adaptation of Euclid's algorithm may be used to efficiently find the integers x and y such that $ax + by = \gcd(a, b)$. The first stage is to run through the steps of Euclid's algorithm, as before. The second stage begins at the second last line of the running of Euclid's algorithm, and involves successive substitution of the lines higher up in the algorithm as illustrated by the following example:

$$\begin{aligned} 65 &= 325 - 2 \times 130 \\ &= 325 - 2 \times (1105 - 3 \times 325) = -2 \times 1105 + 7 \times 325 \\ &= -2 \times 1105 + 7 \times (1430 - 1 \times 1105) = 7 \times 1430 - 9 \times 1105 \\ &= 7 \times 1430 - 9 \times (6825 - 4 \times 1430) = -9 \times 6825 + 37 \times 1430 \end{aligned}$$

That is, $65 = 6825 \times (-9) + 1430 \times 37$, which is the desired representation.

What resources are consumed by Euclid's algorithm? Suppose a and b may be represented as bit strings of at most L bits each. It is clear that none of the divisors k_i or remainders r_i can be more than L bits long, so we may assume that all computations are done in L bit arithmetic. The key observation to make in a resource analysis is that $r_{i+2} \leq r_i/2$. To prove this we consider two cases:

- $r_{i+1} \leq r_i/2$. It is clear that $r_{i+2} \leq r_{i+1}$ so we are done.
- $r_{i+1} > r_i/2$. In this case $r_i = 1 \times r_{i+1} + r_{i+2}$, so $r_{i+2} = r_i - r_{i+1} \leq r_i/2$.

Since $r_{i+2} \leq r_i/2$, it follows that the divide and remainder operation at the heart of Euclid's algorithm need be performed at most $2\lceil \log_2 a \rceil = \mathcal{O}(L)$ times. Each divide-and-remainder operation requires $\mathcal{O}(L^2)$ operations, so that the total cost of Euclid's algorithm is $\mathcal{O}(L^3)$. Finding x and y such that $ax + by = \gcd(a, b)$ incurs a minor additional cost: $\mathcal{O}(L)$ substitutions are performed, at a cost of $\mathcal{O}(L^2)$ per substitutions to do the arithmetic involved, for a total resource cost of $\mathcal{O}(L^3)$.

Euclid's algorithm may also be used to efficiently find multiplicative inverses in modular arithmetic. This is implicit in the proof of corollary; we make it explicit.

Suppose a is co-prime to n , and we wish to find a^{-1} , modulo n . To do so, use Euclid's algorithm and the co-primality of a and n to find integers x and y such that

$$ax + ny = 1$$

Note then that $ax = (1 - ny) = 1(\text{mod } n) = 1(\text{mod } n)$, that is, x is the multiplicative inverse of a , modulo n . Furthermore, this algorithm is computationally efficient, taking only $\mathcal{O}(L^3)$ steps, where L is the length in bits of n .

Now that we know how to efficiently find inverses in modular arithmetic, it is only a short step to solve simple linear equations, such as

$$ax + b = c(\text{mod } n)$$

Suppose a and n are co-prime. Then using Euclid's algorithm we may efficiently find the multiplicative inverse a^{-1} of a , modulo n , and thus the solution to the previous equation,

$$x = a^{-1}(c - b)(\text{mod } n)$$

An important result known as the Chinese remainder theorem extends the range of equations we must solve much further, allowing us to efficiently solve system of equations in modular arithmetic.

Theorem A.7.6. (*Chinese remainder theorem*) Suppose m_1, \dots, m_n are positive integers such that any pair m_i and m_j ($i \neq j$) are co-prime. Then the system of equations

$$\begin{aligned} x &= a_1(\text{mod } m_1) \\ x &= a_2(\text{mod } m_2) \\ &\vdots \\ x &= a_n(\text{mod } m_n) \end{aligned}$$

has a solution. Moreover, any two solutions to this system of equations are equal modulo $M = m_1 m_2 \dots m_n$.

Proof. The proof is to explicitly construct a solution to the system of equations. Define $M_i \equiv M/m_i$, and observe that m_i and M_i are co-prime. It follows that M_i has an inverse modulo m_i , which we denote N_i . Define $x \equiv \sum_i a_i M_i N_i$. To see that x is a solution to the system of equations, note that $M_i N_i = 1(\text{mod } m_i)$ and $M_i N_i = 0(\text{mod } m_j)$ when $i \neq j$, so $x = a_i(\text{mod } m_i)$, which demonstrates the existence of a solution.

Suppose x and x' are both solutions to the system of equations. it follows that $x - x' = 0(\text{mod } m_i)$ for each i , and thus m_i divides $x - x'$ for each i . Since the m_i are co-prime, it follows that the product $M = m_1 \dots m_n$ also divides $x - x'$, so $x = x'(\text{mod } M)$ as we set out to show. \square

Example A.7.10. Consider the following system of linear equations

$$\begin{aligned}x &\equiv 2(\text{mod } 3) \\x &\equiv 3(\text{mod } 5) \\x &\equiv 2(\text{mod } 7)\end{aligned}$$

Thus, here $M = 3 \times 5 \times 7 = 105$. where $m_1 = 3, m_2 = 5, m_3 = 7$. Thus,

$$M_1 = \frac{M}{m_1} = \frac{105}{3} = 35 \quad M_2 = \frac{M}{m_2} = \frac{105}{5} = 21 \quad M_3 = \frac{M}{m_3} = \frac{105}{7} = 15$$

Now finding the inverses of the following linear congruences we get,

$$35N_1 = 1(\text{mod } 3) \quad 21N_2 = 1(\text{mod } 5) \quad 15N_3 = 1(\text{mod } 7)$$

The solutions to the above equations is $N_1 = 2, N_2 = 1, N_3 = 1$. Thus, the solution to the above equation is given as $x = \sum_i a_i M_i N_i$.

$$x = 2 \times 35 \times 2 + 3 \times 21 \times 1 + 2 \times 15 \times 1 = 233$$

Thus, $x = 233(\text{mod } 105) \implies x = 23(\text{mod } 105)$.

Euclid's algorithm and the Chinese remainder theorem are two of the signal triumphs of algorithmic number theory. how ironic that they should play a role in the sequence of ideas leading up to the RSA cryptosystem, whose presumed security is based on the difficulty of performing certain algorithmic tasks in number theory. Nevertheless, this is indeed the case! We turn now to the number-theoretic background necessary to understand the RSA cryptosystem. The key ideas are a famous result of classical number theory, Fermat's little theorem - not to be confused with Fermat's last theorem and a generalization of Fermat's little theorem due to Euler. The proof of Fermat's little theorem relies on the following elegant lemma.

Lemma A.7.7. Suppose p is prime and k is an integer in the range 1 to $\binom{p}{k}$.

Proof. Consider the identity

$$p(p-1) \dots (p-k+1) = \binom{p}{k} k!$$

Since $k \geq 1$ the left hand side (and thus the right) is divisible by p . Since $k \leq p-1$ the term $k!$ is not divisible by p . It follows that $\binom{p}{k}$ must be divisible by p . \square

Theorem A.7.8. (*Fermat's little theorem*) Suppose p is a prime, and a is any integer. Then $a^p = a(\text{mod } p)$. If a is not divisible by p then $a^{p-1} = 1(\text{mod } p)$.

Proof. The second part of the theorem follows from the first, since if a is not divisible by p then a has an inverse modulo p , so $a^{p-1} = a^{-1}a^p = a^{-1}a = 1(\text{mod } p)$. We prove the first part of the theorem for positive a (the case of non-positive a follows easily) by induction on a . When $a = 1$ we have $a^p = 1 = a(\text{mod } p)$, as required. Suppose the result holds true for a , that is, $a^p = a(\text{mod } p)$ and consider the case of $a + 1$. By the binomial expansion,

$$(1 + a)^p = \sum_{k=0}^p \binom{p}{k} a^k$$

By the previous lemma, p divides ${}^p C_k$ whenever $1 \leq k \leq p - 1$, so all the terms except the first and last vanish from the sum modulo p , $(1 + a)^p = (1 + a^p)(\text{mod } p)$. Applying the inductive hypothesis $a^p = a(\text{mod } p)$ we see that $(1 + a)^p = (1 + a)(\text{mod } p)$, as required. \square

There is a remarkable generalization of Fermat's little theorem due to Euler, based on the Euler φ function. $\varphi(n)$ is defined to be the number of positive integers less than n which are co-prime to N . As an example, note that all positive integers less than n which are co-prime to n , and thus $\varphi(p) = p - 1$. The only integers less than p^α which are not co-prime to p^α are the multiples of $p : p, 2p, 3p, \dots, (p^{\alpha-1} - 1)p$, from which we deduce

$$\varphi(p^\alpha) = (p^\alpha - 1) - (p^{\alpha-1} - 1) = p^{\alpha-1}(p - 1)$$

Furthermore, if a and b are co-prime, then the chinese remainder theorem can be used to show that

$$\varphi(ab) = \varphi(a)\varphi(b)$$

To see this, consider the system of equations $x = x_a(\text{mod } a)$, $x = x_b(\text{mod } b)$. Applying the chinese remainder theorem to this set of equations we see that there is a one-to-one correspondence between pairs (x_a, x_b) such that $1 \leq x_a < a$, $1 \leq x_b < b$, $\gcd(x_a, a) = 1$, $\gcd(x_b, b) = 1$, and integers x such that $1 \leq x < ab$, $\gcd(x, ab) = 1$. There are $\varphi(a)\varphi(b)$ such pairs (x_a, x_b) and $\varphi(ab)$ such x .

Thus, together they imply a formula for $\varphi(n)$ based on the prime factorization of n , $n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$:

$$\varphi(n) = \prod_{j=1}^k p_j^{\alpha_j-1} (p_j - 1)$$

Example A.7.11. What is $\varphi(187)$?

Note that the prime factorization of $187 = 11 \times 17$

$$\varphi(187) = \varphi(11)\varphi(17)$$

We know that $\varphi(p) = p - 1$ where p is prime. Thus, $\varphi(11) = 10$ and $\varphi(17) = 16$. Thus, $\varphi(187) = 10 \times 16 = 160$.

Example A.7.12. Prove that

$$n = \sum_{d|n} \varphi(d)$$

where the sum is over all positive divisors d of n , including 1 and n .

Fermat's little theorem has the following beautiful generalization, due to Euler:

Theorem A.7.9. Suppose a is co-prime to n . Then $a^{\varphi(n)} = 1 \pmod{n}$.

Proof. We first show by induction on α that $a^{\varphi(p^\alpha)} = 1 \pmod{p^\alpha}$. For $\alpha = 1$ the result is just Fermat's little theorem. Assume the result is true for $\alpha \geq 1$, so

$$a^{\varphi(p^\alpha)} = 1 + kp^\alpha$$

for some integer k . Then, using $\varphi(p^\alpha) = p^{\alpha-1}(p-1)$,

$$\begin{aligned} a^{\varphi(p^{\alpha+1})} &= a^{p^{\alpha}(p-1)} \\ &= a^{p\varphi(p^\alpha)} \\ &= (1 + kp^\alpha)^p \\ &= 1 + \sum_{j=1}^p \binom{p}{k} k^j p^{j\alpha} \end{aligned}$$

Using the fact that p divides ${}^p C_k$ where $1 \leq k \leq p-1$, it is easy to see that $p^{\alpha+1}$ divides every term in the sum, so

$$a^{\varphi(p^{\alpha+1})} = 1 \pmod{p^{\alpha+1}}$$

which completes the induction. The proof of the theorem is completed by noting that for arbitrary $n = p_1^{\alpha_1} \dots p_m^{\alpha_m}$, $a^{\varphi(n)}$ is a multiple of $\varphi(p_j^{\alpha_j})$. Applying the construction in the proof of the Chinese remainder theorem we find that any solution to the set of equations $x = 1 \pmod{p_j^{\alpha_j}}$. Applying the construction in the proof of the Chinese remainder theorem we find that any solution to the set of equations $x = 1 \pmod{p_j^{\alpha_j}}$ must satisfy $x = 1 \pmod{n}$, and thus $a^{\varphi(n)} = 1 \pmod{n}$. \square

Define Z_n^* to be the set of all elements in Z_n which have inverses modulo n , that is, the set of all elements in Z_n are co-prime to n . Z_n^* is easily seen to form a group of size $\varphi(n)$ under multiplication, that is, it contains multiplicative identity, products of elements in Z_n^* are in Z_n^* , and Z_n^* is closed under the multiplicative inverse operation. For an overview of elementary group theory see the previous section. What is not so obvious is the remarkable structure in Z_n^* has when n is a power of an odd prime p , $n = p^\alpha$. It turns out that $Z_{p^\alpha}^*$ is a cyclic group, that is, there is an element g in $Z_{p^\alpha}^*$ which generates $Z_{p^\alpha}^*$ in the sense that any other element x may be written $x = g^k(\text{mod } n)$ for some non-negative integer k .

Theorem A.7.10. *Let p be an odd prime, α a positive integer. Then $Z_{p^\alpha}^*$ is cyclic.*

A.8 Reduction of factoring to order-finding

The problem of factoring numbers on a classical computer turns out to be equivalent to another problem, the order-finding problem. This equivalence is important as it turns out that quantum computers are able to quickly solve the order-finding problem, and thus can factor quickly. In this section we explain the equivalence between these two problems, focusing on reduction of factoring to order-finding.

Suppose N is a positive integer, and x is co-prime to N , $1 \leq x < N$. The order of x modulo N is defined to be the least positive integer r such that $x^r = 1 \pmod{N}$. The order-finding problem is to determine r given x and N .

Example A.8.1. Show that the order of x modulo N must divide $\varphi(N)$.

We know that from Euler's generalization of Fermat's little theorem that $a^{\varphi(N)} = 1 \pmod{N}$ where a is co-prime to N . Now given that

The reduction of factoring to order finding proceeds in two basic steps. The first step is to show that we can compute a factor of N if we can find a non-trivial solution $x \neq 1 \pmod{N}$ to the equation $x^2 = 1 \pmod{N}$. The second step is to show that a randomly chosen y co-prime to N is quite likely to have an order r which is even, and such that $y^{r/2} \neq \pm 1 \pmod{N}$, and thus $x = y^{r/2} \pmod{N}$ is a solution to $x^2 = 1 \pmod{N}$.

Theorem A.8.1. *Suppose N is a composite number L bits long, and x is a non-trivial solution to the equation $x^2 = 1 \pmod{N}$ in the range $1 \leq x \leq N$, that is, neither $x = 1 \pmod{N}$ nor $x = N - 1 = -1 \pmod{N}$. Then at least one of $\gcd(x - 1, N)$ and $\gcd(x + 1, N)$ is a non-trivial factor of N that can be computed using $\mathcal{O}(L^3)$ operations.*

Proof. Since $x^2 = 1 \pmod{N}$, it must be that N divides $x^2 - 1 = (x+1)(x-1)$, and thus N must have a common factor with one or the other of $(x+1)$ and $(x-1)$. But $1 < x < N-1$ by assumption, so $x-1 < x+1 < N$, from which we see that the common factor cannot be N itself. Using Euclid's algorithm we may compute $\gcd(x-1, N)$ and $\gcd(x+1, N)$ and thus obtain a non-trivial factor of N , using $\mathcal{O}(L^3)$ operations. \square

Lemma A.8.2. *Let p be an odd prime. Let 2^d be the largest power of 2 dividing $\varphi(p^\alpha)$. Then with probability exactly one-half 2^d divides the order modulo p^α of a randomly chosen element of $\mathbf{Z}_{p^\alpha}^*$.*

Proof. Note that $\varphi(p^\alpha) = p^{\alpha-1}(p-1)$ is even, since p is odd, and thus $d \geq 1$. by theorem that there for an odd prime p , α a positive integer, $\mathbf{Z}_{p^\alpha}^*$ is cyclic, there exists a generator g for $\mathbf{Z}_{p^\alpha}^*$, so an arbitrary element may be written in the form $g^k \pmod{p^\alpha}$ for some k in the range 1 through $\varphi(p^\alpha)$. Let r be the order of g^k modulo p^α and consider two cases. the first case when k is odd. from $g^{kr} = 1 \pmod{p^\alpha}$ we deduce that $\varphi(p^\alpha)|kr$, and thus $2^d|r$, since k is odd. The second case is when k is even.

Then

$$g^{k\varphi(p^\alpha)/2} = (g^{\varphi(p^\alpha)})^{k/r} = 1^{k/2} = 1 \pmod{p^\alpha}$$

Thus $r|\varphi(p^\alpha)/2$ from which we deduce that w^d does not divide r .

Summarizing, $\mathbf{Z}_{p^\alpha}^*$ may be partitioned into two sets of equal size; those which may be written g^k with k odd, for which $2^d|r$, where r is the order of g^k , and those which may be written g^k with k even, for which $2^d \nmid r$. Thus, with probability $1/2$ the integer 2^d divides the order r of a randomly chosen element of \mathbf{Z}_p^* , and with probability $1/2$ it does not. \square

A.9 Continued fractions

There are many remarkable connections between the continuum of real numbers and the integers. One such connection is the beautiful theory of continued fractions. In this section we develop a few elements of the theory of continued fractions, elements crucial to the application of the fast quantum algorithms for order-finding and factoring.

As an example of continued fraction, consider the number s defined by the expression

$$s = \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \dots}}}$$

Informally, note that $s = \frac{1}{2+s}$, from which it is easy to satisfy one self that $s = \sqrt{2}-1$. The idea of the continued fractions method is to describe real numbers in terms of integers alone, using expressions such as above. A finite simple continued fraction is defined by a collection a_0, \dots, a_N of positive integers,

$$[a_0, \dots, a_N] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\dots + \cfrac{1}{a_N}}}}$$

We define the n th convergent ($0 \leq n \leq N$) to this continued fraction to be $[a_0, \dots, a_n]$.

Theorem A.9.1. *Suppose x is a rational number greater than or equal to one. Then x has a represented as a continued fraction, $x = [a_0, \dots, a_N]$, which may be found by the continued fractions algorithm.*

Proof. The continued fractions algorithm is best understood by example. Suppose we are trying to decompose $31/13$ as a continued fraction. The first step of the continued fraction is to split $31/13$ into its integer and fractional part,

$$\frac{31}{13} = 2 + \frac{5}{13}$$

Next we invert the fractional part, obtaining

$$\frac{13}{5} = 2 + \frac{1}{\frac{13}{5}}$$

These steps -split the invert are now applied to $13/5$, giving

$$\frac{31}{13} = 2 + \frac{1}{2 + \frac{3}{5}} = 2 + \frac{1}{2 + \frac{1}{\frac{5}{3}}}$$

next we split and invert $5/3$:

$$\frac{31}{13} = 2 + \frac{1}{2 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{2}}}}}$$

The decomposition into a continued fraction now terminates, since $3/2 = 1 + 1/2$ may be written with a 1 in the numerator without any need to invert, giving a final continued fraction representation of $31/13$ as

$$\frac{31}{13} = 2 + \frac{1}{2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{2}}}}$$

It's clear that the continued fractions algorithm terminates after a finite number of split and invert steps for any rational number, since the numerators which appear (31, 23, 2, 1) in the example are strictly decreasing. How quickly does the termination occur? We'll come back to the question shortly. \square

The theorem above has been stated for $x \geq 1$; however, in practice it is convenient to relax the requirement that a_0 to be positive and allow it to be any integer, which results in the restriction $x \geq 1$ becoming superfluous. In particular, if x is in the range 0 through 1 as occurs in applications to quantum algorithms, then the continued fraction expansion has $a_0 = 0$.

The continued fractions algorithm provides an unambiguous method for obtaining a continued fraction expansion of a given rational number. The only possible ambiguity comes at the final stage, because it is possible to split an integer in two ways, either $a_n = a_n$ or $a_n = (a_n - 1) + 1/1$, giving two alternate continued fraction expansions. This ambiguity is actually useful, since it allows us to assume without the loss of generality that the continued fraction expansion of a given rational number has either odd or even number of convergent, as desired.

Example A.9.1. Find the continued fraction expansion for $x = 19/17$ and $x = 77/65$.

$$\frac{19}{17} = 1 + \frac{1}{8 + \frac{1}{2}}$$

and

$$\frac{77}{65} = 1 + \frac{1}{5 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}}}$$

Theorem A.9.2. Let a_0, \dots, a_N be a sequence of positive numbers. Then

$$[a_0, \dots, a_n] = \frac{p_n}{q_n}$$

where p_n and q_n are real numbers defined inductively by $p_0 = a_0, q_0 = 1$ and $p_1 = 1 + a_0 a_1, q_1 = a_1$, and for $2 \leq n \leq N$,

$$\begin{aligned} p_n &= a_n p_{n-1} + p_{n-2} \\ q_n &= a_n q_{n-1} + q_{n-2} \end{aligned}$$

In the case where a_j are positive integers, so too are the p_j and q_j .

Proof. We induct on n . the result is easily checked directly for the cases $n = 0, n = 1, 2$. By definition, for $n \geq 3$.

$$[a_0, \dots, a_n] = [a_0, \dots, a_{n-2}, a_{n-1} + 1/a_n]$$

Applying the inductive hypothesis, let \tilde{p}_j/\tilde{q}_j be the sequence of convergents associated with the continued fraction on the right hand side:

$$[a_0, \dots, a_{n-2}, a_{n-1} + 1/a_n] = \frac{\tilde{p}_{n-1}}{\tilde{q}_{n-1}}$$

It is clear that $\tilde{p}_{n-3} = p_{n-3}, \tilde{p}_{n-2} = p_{n-2}$ and $\tilde{q}_{n-2} = q_{n-2}$, so

□

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

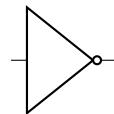
B.1.1 NOT Gate

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 . It's symbol is as shown:



Mathematically, the NOT gate is defined as:

$$f(x) = \neg x$$

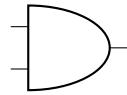
B.1.2 AND Gate

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: The AND gate is denoted by the symbol \wedge . It's symbol is as

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

shown:



Mathematically, the AND gate is defined as:

$$f(x, y) = x \wedge y$$

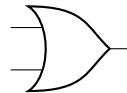
B.1.3 OR Gate

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: The OR gate is denoted by the symbol \vee . It's symbol

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

is as shown:

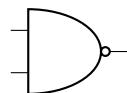


Mathematically, the OR gate is defined as:

$$f(x, y) = x \vee y$$

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: The NAND gate is denoted by the symbol \neg . It's symbol is as shown:



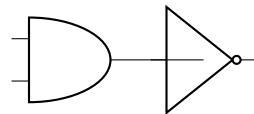
Input 1	Input 2	Output
0	0	1
0	1	1
1	0	1
1	1	0

Table B.4: Truth table for NAND Gate

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:

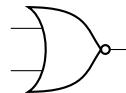


B.1.5 NOR Gate

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: The NOR gate is denoted by the symbol $\overline{A \vee B}$. Its symbol is as shown:

Input 1	Input 2	Output
0	0	1
0	1	0
1	0	0
1	1	0

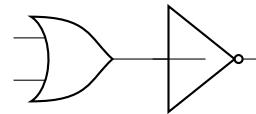
Table B.5: Truth table for NOR Gate



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:



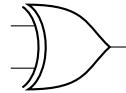
B.1.6 XOR Gate

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: The XOR gate is denoted by the symbol \oplus . It's symbol

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

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: Mathematically, the FANOUT gate is defined as:

$$f(x) = (x, x)$$

Input	Output 1	Output 2
0	0	0
1	1	1

Table B.7: Truth table for FANOUT Gate

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

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 .*

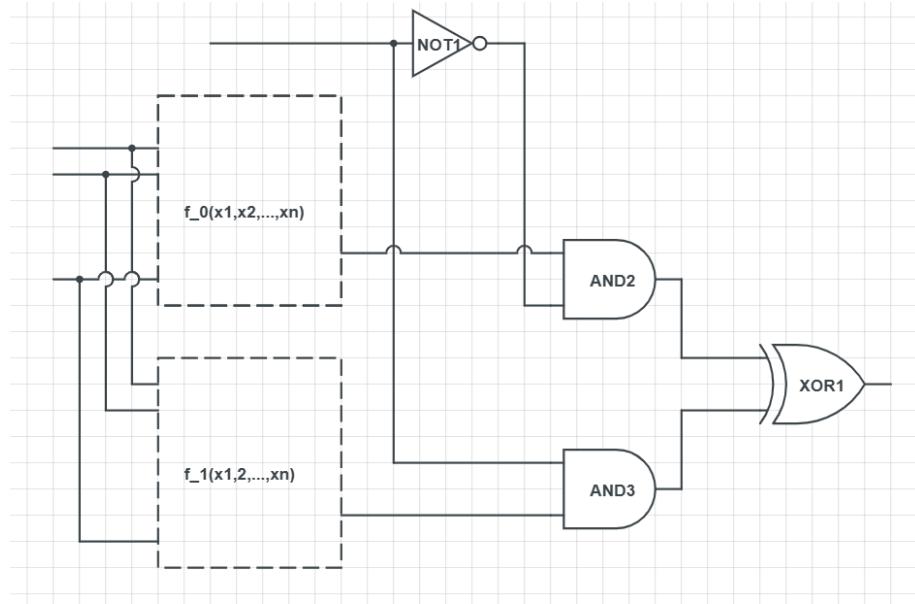
Proof. Any fixed gates can be used to compute any function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$. Consider the proof for the simplified function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ with n input bits and a single output bit. This is a Boolean function. Using mathematical induction, For $n = 1$, there 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. \square

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 bits 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.

Figure B.1: Circuit to compute f

Example B.2.1. Half adder and Full adder circuits used in Classical Computers.

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.

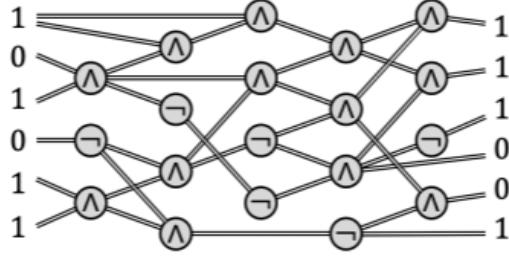


Figure B.2: Generic form of a classical circuit (information flows from left to right)

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 a 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 randomize 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:

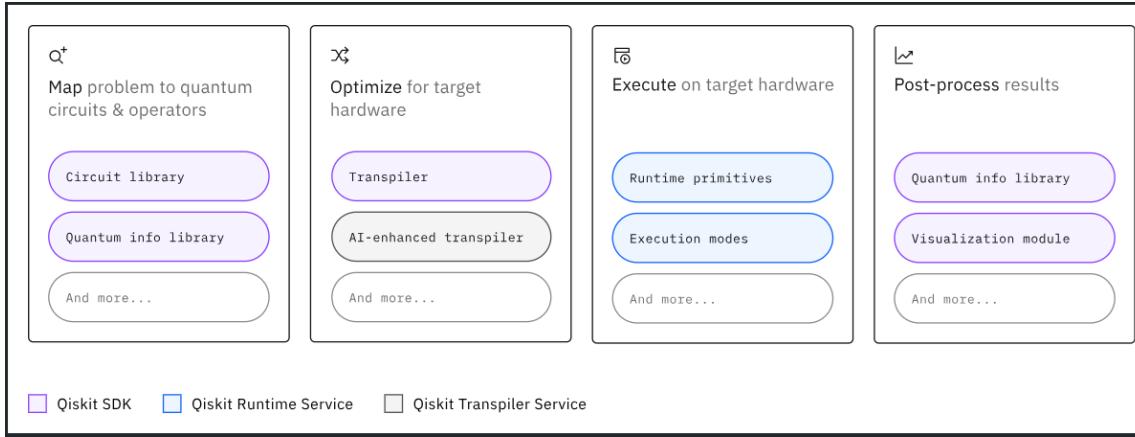


Figure C.1: Qiskit Steps

- 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 analyze 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

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 and 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 life cycle management of quantum jobs- from defining program specifications to job submission through to the post-processing 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 impact 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 separate 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 quantum 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.

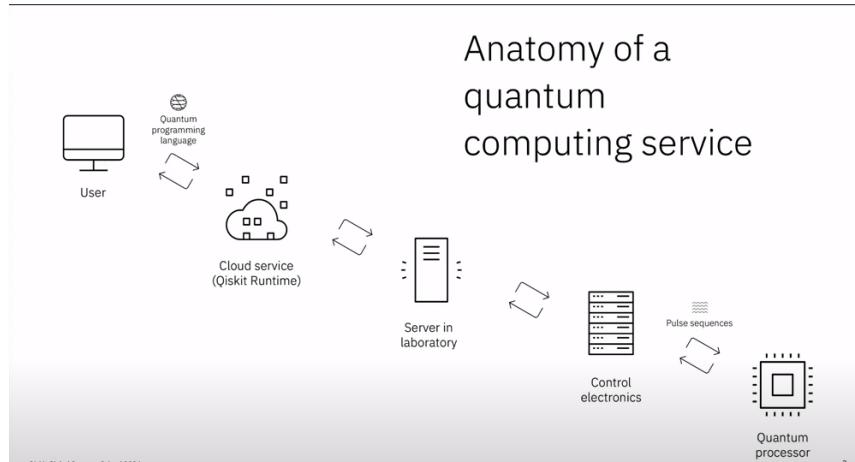


Figure C.2: Anatomy of Quantum Computing Service

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

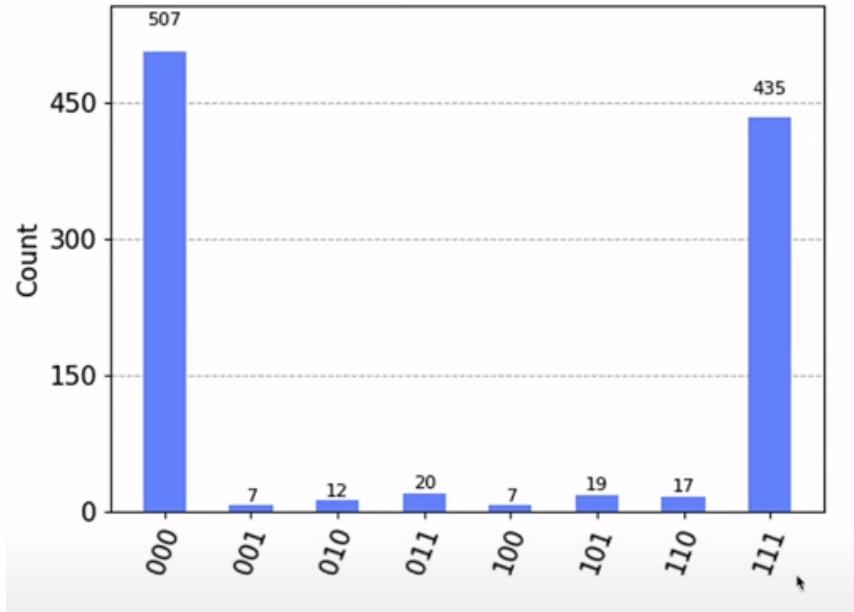


Figure C.3: Example of final result

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 bit strings 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.