

# NOTES ON QUANTUM COMPUTATION AND INFORMATION

Raghav G. Jha  <sup>1</sup>

*Thomas Jefferson National Accelerator Facility, Newport News, VA 23606, USA  
Perimeter Institute for Theoretical Physics, Waterloo, Ontario N2L 2Y5*

---

## Abstract

We discuss fundamentals of quantum computing and information - quantum gates, circuits, algorithms, theorems, error correction, and provide collection of QISKIT<sup>2</sup> programs and exercises for the interested reader.

---

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Notation and all that</b>	<b>7</b>
<b>3</b>	<b>States, quantum gates, density matrices, and rules</b>	<b>10</b>
3.1	States on the Bloch sphere and gates . . . . .	10
3.2	General quantum operations . . . . .	20
3.3	Entanglement . . . . .	22
3.3.1	Partial trace method . . . . .	24
3.3.2	Entangled mixed states . . . . .	25
3.4	Quantum Fourier Transform (QFT) . . . . .	26
3.5	Quantum rules and limitations . . . . .	27

---

<sup>1</sup>[raghav.govind.jha@gmail.com](mailto:raghav.govind.jha@gmail.com)

<sup>2</sup>It is an open source software development kit (SDK) for working with OpenQASM and the IBM Q quantum processors. Various IBM quantum simulators can be found [here](#)

<b>4</b>	<b>Towards Quantum algorithms</b>	<b>29</b>
4.1	Full-adder . . . . .	29
4.2	Phase kickback . . . . .	30
4.3	Deutsch-Jozsa Algorithm . . . . .	30
4.4	Bernstein-Vazirani algorithm . . . . .	32
4.5	Grover’s algorithm . . . . .	32
4.6	Quantum phase estimation (QPE) algorithm . . . . .	33
4.7	Shor’s factoring algorithm . . . . .	34
<b>5</b>	<b>Variational Quantum Eigensolver (VQE)</b>	<b>36</b>
5.1	Quantum Anharmonic Oscillator . . . . .	38
5.2	$O(3)$ non-linear sigma model . . . . .	39
<b>6</b>	<b>Quantum Error Correction (QEC)</b>	<b>41</b>
6.1	Bit-flip error . . . . .	42
6.2	Phase-flip error . . . . .	43
6.3	Shor’s nine qubit code . . . . .	44
<b>7</b>	<b>Looking at NISQ and beyond</b>	<b>46</b>
<b>8</b>	<b>Future</b>	<b>48</b>
<b>9</b>	<b>Acknowledgements</b>	<b>49</b>
<b>A</b>	<b>Some examples using Qiskit</b>	<b>49</b>
A.1	Installing package and quantum hello world . . . . .	51
A.2	Some basic operations using qubits and gates . . . . .	52
A.3	Quantum full-adder . . . . .	55
A.4	Grover’s algorithm with general diffuser routine . . . . .	56
A.5	Implementing QFT . . . . .	57
A.6	Bit-flip code, Phase kickback . . . . .	59
A.7	VQE solution to anharmonic quantum oscillator for three qubits . . . . .	61
A.8	Phase estimation . . . . .	65
A.9	Sample MATHEMATICA code for to compute vN entropy . . . . .	66
	<b>References</b>	<b>67</b>

## Introduction

---

One of the quotes that profoundly expresses the present state of the development in quantum computation and information is by Wheeler. In his colourful language, he said - ‘I think of my lifetime in Physics as divided into three periods. In the first period, I was in the grip of the idea that everything is particles. I call my second period, everything is fields, and now I am in the grip of a new vision that everything is information’. In fact, we can safely add ‘quantum’ to this since it appears that everything is ‘quantum information’. There are several physicists who have explored this line of thought, see for example Refs. [1–3] and references therein to start a trail.<sup>3</sup>

The goal of theoretical physics is to understand the principles that govern the universe at different length scales. It is a challenging problem since the associated scales vary over a wide range (about thirty orders of magnitude) from inside the nucleus ( $10^{-15}$  m) to the size of typical galaxies ( $10^{15}$  m). We have two extremely accurate theories that govern each of them separately. The incredible progress made in the last 125 years has given us the basic principles of quantum mechanics and special theory of relativity which when combined into the framework of ‘quantum field theory’ (QFT) holds the key in explaining a major portion of all physical phenomena occurring in nature at sub-nuclear scales and culminating in the Standard Model of particle physics. Some jewels of this framework are - calculating the magnetic moment of the electron to about ten decimal places, and the discovery of the Higgs boson at the Large Hadron Collider (LHC) at CERN. On the other hand, at the other extreme of the spectrum, we have the highly successful theory of general relativity which describes classical gravity, predicts the existence of black holes and has been tested to a great accuracy culminating with the discovery of gravitational waves.

Another topic close to physicists is that of computing and since computing is a physical process it is governed by the laws of Physics. Computers<sup>4</sup> have changed and affected every facet of life around us. From handling complex machines like airplanes, to the RSA (Rivest–Shamir–Adleman) key protocol, which forms the basis of all internet transactions. But there are certain classes of problems where even the fastest supercomputers of today’s era fail. They fail because we have to

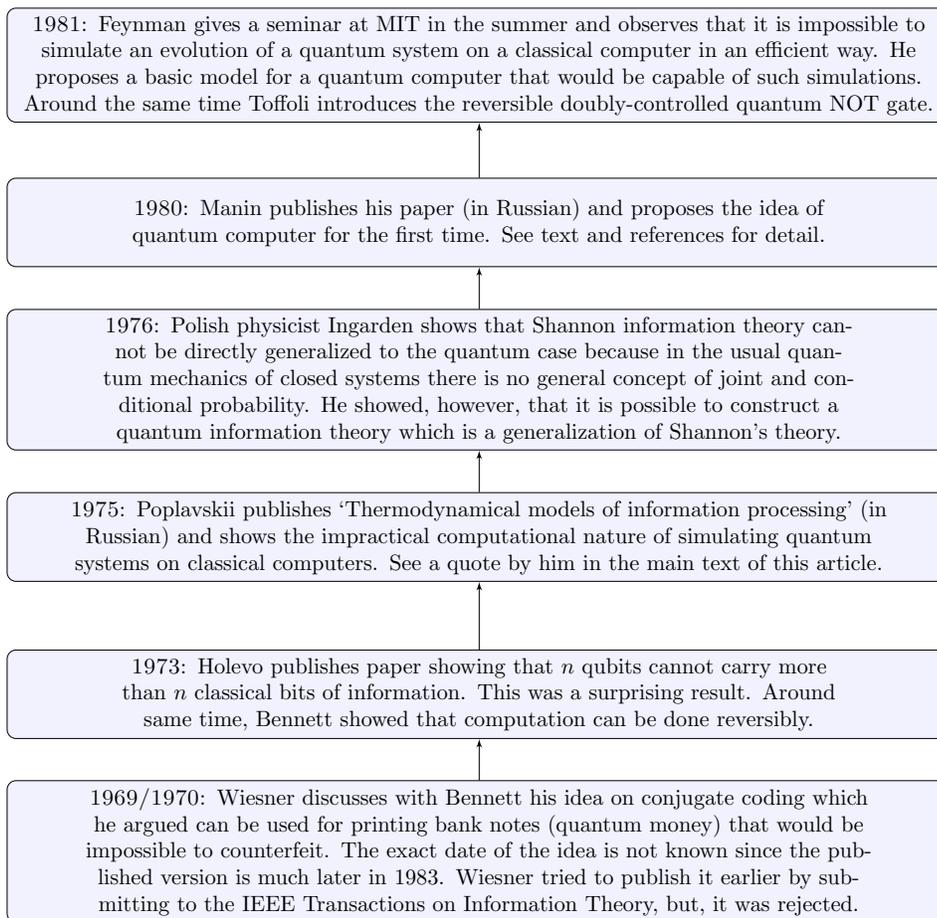
---

<sup>3</sup>The quintessential example of how crucial information is can be understood from a well-known paradox due to Maxwell (1867). This is known as ‘Maxwell’s demon’. This thought experiment is an apparent violation of the second law of thermodynamics. Szilard wrote an interesting paper in 1929 which took a big step towards the resolution, but it did not ‘totally’ explain the paradox. It was Landauer’s work [4] and Bennett’s follow-up [5] that was resolved after 115 years and was one of the major problems that raged physicists through the century. The solution lies in the fact that information is physical, just like the entropy or energy is in thermodynamics. Additionally, it is interesting to note that von Neumann in his letter to Teller on April 8, 1947 wrote - “I have some definite ideas concerning automata, memory, ‘clearing’ and irreversibility and entropy, somewhat connected to Szilard’s treatment of Maxwell’s demon and my old treatment of entropy and observability in quantum mechanics” [p. 245, John von Neumann Selected Letters, American Mathematical Society (October 1, 2005)] but we believe it was never written up by either of the Martians. The statement appears to the author to be closely related to the idea of erasure and non-violation of second law as established by Landauer et al.

<sup>4</sup>Computer is a machine that maps an array of bits (0 and 1) from one configuration to another. If it manipulates quantum bits, it is called a quantum computer.

wait for thousands of years to solve a problem using classical ideas of bits, on which they are based. Though, classical computation has rapidly progressed to the extent that we can now have a billion transistors in a single laptop, it is probably not possible that size of these transistors can be made smaller than size of say hydrogen atom which is about  $10^{-10}$  m or the frequency of the clock speed can be increased more than 1 PHz ( $10^{15}$  Hz) which is roughly the frequency of atomic transitions. So, it is clear that supercomputers will soon exhaust their capability and problems with exponential complexity won't be solved using them. This leads to a paradigm shift in our idea of computing. Along this direction, already in the 1970s, Benioff started thinking about theoretical feasibility of quantum computing and his paper [6, 7] which described a quantum mechanical model of Turing Machines was one of the first papers in the field of quantum computing. This work was based on a classical description in 1973 of reversible Turing machines from Ref. [8]. In the same year, there was another work [9] by Russian mathematician Manin who argued that the superposition and entanglement features of quantum mechanics would make for a substantial speedup compared to classical methods. Soon after, Feynman [10] argued that quantum computers will be better suited to these tasks than any classical computer we can ever build because they can store much more information compared to classical computers. He pointed out that - 'Let the computer itself be built of quantum mechanical elements which obey quantum mechanical laws'. In the early days of quantum computing, Poplavskii argued [11] - "The quantum-mechanical computation of one molecule of methane requires about  $10^{42}$  grid points. Assuming that at each point we have to perform only 10 elementary operations, and assume that those calculations are performed at temperatures of 0.003 Kelvin, we would still have to use all the energy produced on Earth during the last century". This makes it clear that a full-fledged computation of quantum dynamics is beyond the reach of any classical computer. Nature is quantum mechanical and hence we have to make use of it. These problems and ideas gave rise to the field of quantum computation. In summary, quantum computation addresses one of the two issues that limit any classical computation: space (memory) and time. In quantum computers, we can store exponentially large information, so it is memory wise way more efficient than classical counterpart. We provide the first twelve years of the timeline of quantum computing in Fig. 1.

The usefulness of classical computers has been a monumental achievement of human mankind in the past century. The basic fundamental idea of classical computers (or computing) is the binary two-state signal: 0 and 1. It is practically the only thing a computer understands at its most fundamental level. A computer practically does two things: 1) it stores information as a string of 0s and 1s (also known as classical bits), and 2) Transform them based on instructions given. It achieves the second task by using gates which perform simple logic operations to transform bits. Some examples of such gates are AND, NOT, OR gates. However, we know that at the microscopic level, the laws of Physics and hence nature is purely quantum-mechanical. It marks a paradigm shift in our understanding compared to the classical world, which is deterministic. In simple terms, this means that while a classical bit can at a given time be either 0 or 1, a quantum bit (known as 'qubit') can be in a superposition of these two options and hence we can never conclude exactly which one until we measure it. The existing quantum devices contain  $\mathcal{O}(100)$  qubits but they are



**Figure 1.** The brief history of quantum computation.

noisy, so actual efficiency might be a small fraction of it. They are sometimes referred to as ‘Noisy Intermediate-Scale Quantum (NISQ)’ devices also sometimes referred to as ‘NISQ era’.

To understand the power of principles of quantum mechanics, consider the amount of digital data in the world. It is estimated that the total digital data in the world by 2025 will be close to 175 ZB (1 ZB =  $10^{21}$  bytes). This data can be contained in total if we have about 78 fully efficient qubits. This is the power of qubits (growth of ‘power of 2’).<sup>5</sup> However, such power often comes

<sup>5</sup>Once a king offered any reward to a man who had done good for the kingdom. The man asked that a single grain of rice be placed on the first square of the chessboard. Then two grains on the second square, four grains on the third, and so on and doubling each time. The king was taken back by such a small request and agreed. He ordered the treasurer to pay the sum agreed upon. After few days, the treasurer explained that the sum could not be paid since by the time he got halfway through the board, the amount of grain required was more than the entire kingdom possessed. This is the exponential growth known to kings several thousand years before. I first heard this story from my grandfather when I was 8, however, this is now easily found online and also appear in some popular science books.

with its own set of troubles and limitations. For a review of quantum algorithms and other details, see Ref. [12]. As of today, there exists only a handful of useful<sup>6</sup> quantum algorithms, and it is not yet known how many classes of problems will eventually benefit by exploiting the quantum features. One of the major applications of quantum computing is in the field of quantum many-body systems. The mightiest of computers today in the world cannot deal fully (without approximation and problems) with even a quantum system of highly interacting 100 electrons! Therefore, solving this is one of the major challenges in theoretical and computational Physics. Even though we only have a relatively small ( $< 100$ ) number of non-error corrected qubits and the implementation faces several problems, it is expected that in the coming 2-3 decades the situation will change drastically. Even 100-200 error-corrected qubits in the future can completely revolutionise the nuclear computations and whole of quantum chemistry and several areas of Physics.

Though, we will mostly restrict to the theoretical side of quantum computing in these notes, it is good to point out some basic ideas about the experimental side of this story. The act of fabricating a proper logical qubit is a challenging task. In order to actually realize a quantum computer, we need to have qubits which can retain their properties and those which can be made to evolve/change as computationally desired. We must also prepare qubits in some initial state and measure them conveniently. In order to construct a computer as envisaged by Manin and others with these properties, DiVincenzo in 2000 [13] listed some conditions necessary to achieve this based on his attempts to construct a quantum computer (a machine which can efficiently simulate quantum systems such as solving the quantum many-body problems). These conditions are now known as ‘DiVincenzo criteria’ and mentioned below:

- A scalable physical system with well-characterised qubits.
- The ability to initialise to some fiducial state such as  $|000000 \dots 00\rangle$ .
- Long decoherence time compared to gate operation time ( $\tau_{\text{deco.}} \gg \tau_{\text{op.}}$ ).
- A universal set of quantum gates.
- A qubit specific measurement capability.

These conditions are for quantum computation, such as superconducting quantum computing or trapped ion approaches. There are two more conditions which apply to quantum communication. In order to understand these two conditions, we have to explain stationary and flying qubits. Stationary qubits are those which are in the computer and on which we apply gates, while flying qubits are ones that move (say, like photons). With these definitions, the last two conditions are:

- The ability to interchange between stationary and flying qubits<sup>7</sup>.
- The ability to faithfully transmit flying qubits between specified locations.

---

<sup>6</sup>Polynomial or exponential speedup compared to best-known classical algorithms

<sup>7</sup>Flying qubits have restricted use compared to the stationary qubits. They are only used to pass the information over some distance, whereas stationary qubits have a two fold task: store information and perform calculations.

The experimental challenge is that these conditions can often only be partially met. Single photons have two polarization states and are almost ideal qubits, but the optical material required to make them interact is difficult in practice without loss of coherence. One can represent spin-1/2 particles as qubits as is often done theoretically and experimentally for the trapped ion method to quantum computing, but the phonons which are used to mediate the interactions have short coherence time. Another approach is to use electric charge as a qubit representation and use the excellent techniques from modern electronics, but even here lies the problem of decoherence. One advanced approach is to use charge carriers in not a usual metal, but in a superconductor. The superconductor qubit representation is favoured due to robustness of the Cooper pairs involved. In fact, there are by now some advancements achieved over this as well. As of today's technology, we have some major categories for these constructions. The *first* is the 'superconducting qubits' (SC). It was first observed that Josephson junction (JJ)<sup>8</sup> is the element that provides the condition to turn a superconducting circuit into a qubit. One example is Google's Sycamore 53-qubit chip, which belongs to this category. In fact, these quantum processors use an advanced version of this which has more than one JJ and is known as 'transmons' which is the shortened name for 'transmission line shunted plasma oscillation qubit'. Both IBM and Google quantum processors use these modified qubit approach. It is closely related to the charge qubit (cooper pair box) but has better effectiveness due to improvement in decoherence time. There is *another* approach known as 'topological qubits' (Microsoft, Bell Labs) which underlies the application of Majorana fermions to create qubit states. There are other technologies based on trapped ions, silicon dots (Intel), and diamond vacancies. We will not discuss these here, since it would really digress us from the goal of this article.

In these notes, the emphasis is to introduce the idea of quantum computing inspired by the connection to classical computing and then use some SDKs to perform some simple computation on IBM's backend machines. These are not meant to be exhaustive, and the purpose is not to solve some model in Physics which cannot be solved in a reasonable time using computers we have today, but to introduce the elements which might in coming decades be part of advanced quantum algorithms and computation that can achieve those dreams. However, we want to make it clear to the reader that even with quantum computing and reliable error-correction some physical problems might still not be solved in polynomial time. In order to understand this issue, one must talk in terms of complexity classes i.e., the problem of how the computational resources scale with increasing system size. This is one of the main problems in computer science. We say that  $\mathbf{P}$  is the set of problems where time scales as polynomial in system size on a deterministic Turing machine. This is also sometimes referred to as a problem that can be solved<sup>9</sup> efficiently. The classical complexity classes introduced for Turing machines of both kinds (deterministic or probabilistic)

---

<sup>8</sup>A Josephson junction is made of a non-superconducting material between two layers of superconducting electrodes. A Cooper pair of electrons can tunnel through the non-superconducting barrier from one superconductor to another.

<sup>9</sup>It is interesting to note that Gödel's letter to von Neumann in 1956 (when latter was in the hospital) discussed a function  $\phi$  which has some relation to complexity classes. It is probably the first written instance of  $\mathbf{P}$  versus  $\mathbf{NP}$  issue. However, since von Neumann died early 1957, his answer to this letter was never known.

have since then been extended to admit quantum complexity classes. For example, **BQP** stands for bounded-error quantum polynomial time and is referred to the class of decision problems solvable by a quantum computer in polynomial time. It is the quantum analogue of the complexity class **BPP**. But since a quantum circuit can simulate a classical one, both **P**, and its probabilistic version **BPP**, are contained in **BQP**. It is known that the scattering problem in interacting QFT lies in **BQP**. In fact, not all physical problems lie in **BPP** or **BQP**, some of them belong to **NP** which stands for *non-deterministic polynomial time* requiring more than polynomial resources assuming that  $\mathbf{P} \neq \mathbf{NP}$ <sup>10</sup> but whose solution can be verified in polynomial time. The quantum analogue of **NP** with polynomial intractable problems is denoted by **QMA**<sup>11</sup>, and problems that belong to this class are *unlikely* to be solved efficiently even with a quantum computer assuming  $\mathbf{BQP} \neq \mathbf{QMA}$ . **QMA**-hard (defined similar to **NP**-hard), is a class of problems if every problem in **QMA** can be reduced to it. A problem is said to be **QMA**-complete if it is **QMA**-hard and in **QMA**. One of the motivations for quantum computers is the potential to solve interesting problems which are **NP** hard like the model with sign problems. These cannot be efficiently solved on classical computers. However, some of these might still not belong to **BQP** and therefore might still be inaccessible with quantum resources. For example, it was shown in Ref. [14] that scattering in scalar QFT is **BQP**-complete, indicating that all problems in **BQP** can be mapped to scattering in scalar QFT with polynomial scaling time to solution, and further that the scattering problem itself is in **BQP** and thus can be efficiently simulated quantum mechanically. We say a problem is **BQP**-complete when it is both **BQP** and **BQP**-hard. Similarly, for classical version, this means that a problem is said to be **NP**-hard if everything in **NP** can be transformed in polynomial time into it even though it may not be in **NP**. Conversely, a problem is **NP**-complete if it is both in **NP** and **NP**-hard.

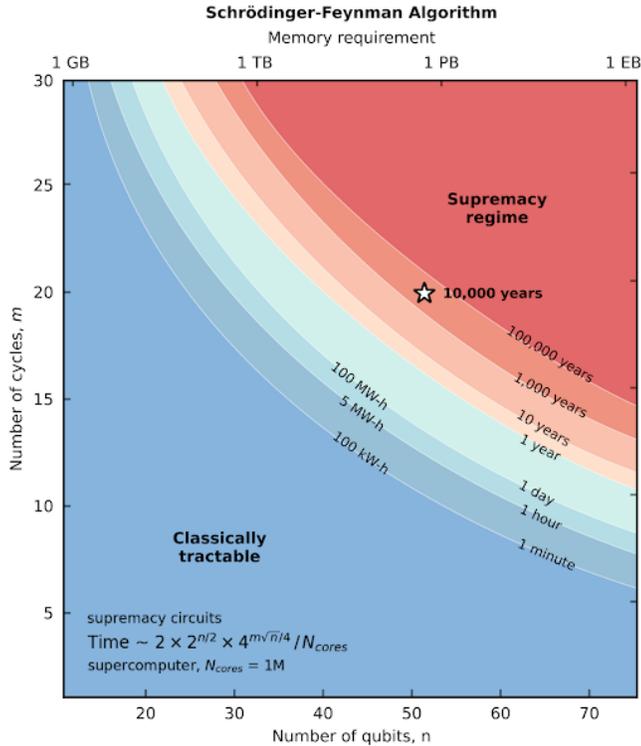
We now give an outline of the notes. In Sec. 2, we explain the notation and basic fundamentals. In the next section, we introduce the idea of quantum states, quantum logic gates, density matrices, and the important quantum Fourier transform method. In Sec. 4, we mention several quantum algorithms such as phase kickback, Deutsch algorithm, Grover's search algorithm, Kitaev's phase estimation, and Shor's algorithm. In Sec. 5, we discuss the variational quantum eigensolver (VQE) method to solve some simple problems. In Sec. 6, we mention an important feature of quantum computing i.e., quantum error correction and explain bit and phase flip and universal code to correct arbitrary (phase or flip) single qubit errors. In the Appendix, i.e., Sec. A, we provide codes and instructions on how the user can run some simple simulations on a web browser (Google Collaboratory).

## SECTION 2

### Notation and all that

<sup>10</sup> $\mathbf{P} \neq \mathbf{NP}$  has enormous practical and economic importance, because modern cryptography is based on assumption that they are not equal. This means that there exist problems that are impossible for computers to solve, but for which the solutions are easily checked (polynomial). A failure of this could create havoc in modern cryptography resulting in breakdown of secure transactions all over the world.

<sup>11</sup>QMA is an acronym for Quantum Merlin Arthur



**Figure 2.** Estimate of the equivalent classical computation time assuming 1M CPU cores for quantum supremacy circuits as a function of the number of qubits and number of cycles for the Schrödinger-Feynman algorithm. The star shows the estimated computation time for the largest experimental circuits. To know more about Schrödinger-Feynman (SF) algorithm, please see Ref. [15] and references therein. In short, the Schrödinger’s method takes about  $m^{2^n}$  time and about  $2n$  memory, while Feynman’s method takes about  $4^m$  time and about  $m + n$  memory. Feynman’s algorithm saves memory because it calculates an amplitude instead of keeping track of the state vector throughout like in the Schrödinger approach. However, the fact that more intermediate steps are to be done means more time. There are some methods which combine features of both, and those belong to the class of SF algorithms. This figure is taken from Ref. [16]. This diagram should be taken with a grain of salt and only meant to provide a basic idea.

Before we delve into describing the notations and discussing basic ideas, we emphasize and point out the differences between digital classical and quantum computing in Table 1. We will restrict to the well-known Dirac’s<sup>12</sup> bra-ket notation[17] which deals with states in Hilbert space based on von Neumann abstract and mathematical precise formulation of quantum mechanics written down around 1932<sup>13</sup>. We will briefly review this for interested readers. Suppose we have a column and

<sup>12</sup>Probably inspired by Grassmann’s similar notation used several decades before

<sup>13</sup>It might be surprising that just about three years later in 1935, he wrote to Birkhoff from 1935 – ‘I would like to make a confession which may seem immoral: I do not believe in Hilbert space anymore’. We encourage the interested reader to explore this direction if interested.

Elements	Classical	Quantum
Algebra	Boolean	Linear
Basic Unit	Bit	Qubit
Gates	Logic gates	Unitary gates
Reversibility of gates	Sometimes	Always
Example of universal gate set	{NAND}	{H, T, CNOT}
Correction of errors	Repetition	Shor-like code

**Table 1.** Comparison between classical and quantum digital computing.

row vector as given below:

$$|B\rangle = \begin{pmatrix} B_1 \\ \vdots \\ \vdots \\ B_N \end{pmatrix}, \quad \langle A| = (A_1^* \cdot \cdot \cdot \cdot A_N^*) \quad (2.1)$$

We can do two operations using them by computing the inner product, i.e.,  $\langle A|B\rangle$  or the outer product denoted by  $|A\rangle|B\rangle$ . The first returns a number, while the latter gives a matrix. The row vector is called ‘bra’ while the column vector is called ‘ket’ and they are both dual to each other (in vector space sense). This state denoted as ‘ket’ is postulated to contain all the information about the physical state which we wish to know. We have the following relations:  $\langle A|^\dagger = |A\rangle$ , and  $|A\rangle^\dagger = \langle A|$ . In quantum mechanics, the state of a system is described by a ray  $|\psi\rangle$  in Hilbert space,  $\mathcal{H}$  where the ray has unit norm. If we have,  $\lambda = e^{i\phi}$  then we see that  $|\psi\rangle$  and  $\lambda|\psi\rangle$  both have the same norm because  $\lambda$  is just a phase, and they represent the same physical state. The phase of the ray is not an observable. The rays of Hilbert space  $\mathcal{H}$  are the equivalence class of unit vector that only differ by a phase. Two states represented by ‘kets’ can be added together as:  $|\alpha\rangle + |\beta\rangle = |\gamma\rangle$ . In fact, if  $|\psi_0\rangle$  and  $|\psi_1\rangle$  are two orthogonal states, then their linear superposition ( $c_1|\psi_0\rangle + c_2|\psi_1\rangle$ ) is also a well-defined state but must satisfy  $|c_1|^2 + |c_2|^2 = 1$ . We refer to  $c_1$  and  $c_2$  as ‘amplitudes’ corresponding to  $|\psi_0\rangle$  and  $|\psi_1\rangle$  respectively. We can multiply a ket by a complex number  $c$  as:  $|\alpha\rangle = c|\psi\rangle$  or  $|\psi\rangle c$ , however, physically speaking there is no difference between  $|\alpha\rangle$  and  $|\psi\rangle$  as discussed above. When we want to measure an observable, we apply the corresponding operator from the left on the ket as:  $A \cdot |\psi\rangle$  or  $\hat{A}|\psi\rangle$ . Two kets denoted by  $\alpha$  and  $\beta$  are said to be orthogonal if:  $\langle \alpha|\beta\rangle = \langle \alpha|\beta\rangle = \langle \beta|\alpha\rangle^* = 0$ . There are also some operations which are not allowed. The operator must be either on the left of ket or on the right of a bra, i.e.,  $|\psi\rangle A, A \langle \psi|$  are both incorrect for an operator  $A$ . Some products like  $|\alpha\rangle|\beta\rangle$  are not allowed if the ket vectors belong to same vector space. Note that when we write  $|01\rangle = |0\rangle|1\rangle$  later on, we mean vector spaces of two distinct qubits so it is allowed. The expectation value of any observable corresponding to the Hermitian operator  $A$  is given by:  $\langle A\rangle = \langle \alpha|\hat{A}|\alpha\rangle$ . In fact, we can also write this as:

$$\langle A \rangle = \sum_{a', a''} \langle \alpha | a'' \rangle \langle a'' | A | a' \rangle \langle a' | \alpha \rangle = \sum_{a'} a' |\langle a' | \alpha \rangle|^2$$

Consider a spin-1/2 particle with spin-up  $|\uparrow\rangle$  or down  $|\downarrow\rangle$ . The Hilbert space in this case is two-dimensional with two orthonormal basis which we denote as  $|0\rangle$  and  $|1\rangle$ . This is an example of quantum bit ('qubit'). One often refers to the orthonormal basis  $\{|0\rangle, |1\rangle\}$  as computational basis. Any state  $|\alpha\rangle$  can be decomposed as a convex combination of projectors onto pure states <sup>14</sup> i.e.,  $|\alpha\rangle = \sum_i p_i |\psi_i\rangle \langle \psi_i | \alpha \rangle$ . In quantum computation, we have the following standard definitions:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \quad (2.2)$$

These are one-qubit state. We can have two-qubit states as well, and then we use shorthand like

$$|00\rangle = |0\rangle |0\rangle = |0\rangle \otimes |0\rangle, \quad |10\rangle = |1\rangle |0\rangle \text{ and so on. Therefore we have, } |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \text{ and}$$

so on. We refer to a state as 'normalized' if  $\langle \psi | \psi \rangle = 1$ . This is called the inner product of a 'bra' and 'ket'. For readers comfortable with this notation, nothing remains to be said, but for those who are new, we give a small example of 'bra-ket' gymnastics now. Suppose we have a system in state  $|i\rangle$  and another in  $|j\rangle$ , then the composite state is written as  $|ij\rangle = |i\rangle |j\rangle$ . For example,  $\langle j | \langle i | k \rangle | l \rangle = |ij\rangle^\dagger |kl\rangle = \langle j | \langle i | k \rangle | l \rangle = \langle i | k \rangle \langle j | l \rangle$ . One can also define an 'outer product' as,  $|\psi\rangle \langle \phi|$  which is a matrix. Suppose we take  $|0\rangle, |1\rangle$  and construct  $\rho = 1/2(|0\rangle \langle 0| - |0\rangle \langle 1| - |1\rangle \langle 0| + |1\rangle \langle 1|)$ . The matrix  $\rho$  is given by:

$$\rho = \frac{1}{2} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}.$$

More systematically, we define the outer product as:

$$|\psi\rangle \langle \phi| = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} (\gamma^* \ \delta^*) = \begin{pmatrix} \alpha\gamma^* & \alpha\delta^* \\ \beta\gamma^* & \beta\delta^* \end{pmatrix}. \quad (2.3)$$

In fact, any square matrix can be written as linear combination of outer products. The simplest example is to write any  $2 \times 2$  matrix in this form given by:

$$A = A_{00} |0\rangle \langle 0| + A_{01} |0\rangle \langle 1| + A_{10} |1\rangle \langle 0| + A_{11} |1\rangle \langle 1| \quad (2.4)$$

This representation of a matrix in terms of 'ket-bra' also provides simple interpretation of some quantum gates. For example,  $|00\rangle \langle 00| + |01\rangle \langle 01| + |10\rangle \langle 11| + |11\rangle \langle 10|$  gives a representation of CNOT gate<sup>15</sup> and implies the corresponding transformation.

<sup>14</sup>Often mentioned as 'inserting an identity'

<sup>15</sup>We will soon learn about this gate in the next sections

EXAMPLE 1: Consider  $|\psi\rangle = |a\rangle \otimes |b\rangle$ ,  $A|a\rangle = a|a\rangle$ , and  $B|b\rangle = b|b\rangle$ . Compute  $A \otimes B|\psi\rangle$ .

□ We can write  $A \otimes B|\psi\rangle$  as  $A \otimes B|ab\rangle = (A \otimes B)|a\rangle \otimes |b\rangle$  and then by distributing the operators write this as  $A|a\rangle \otimes B|b\rangle = ab|\psi\rangle$ .

SECTION 3

## States, quantum gates, density matrices, and rules

---

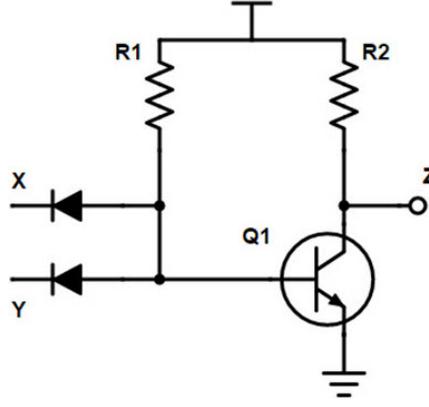
In classical logic gates, the input signal can be a string of either 1 or 0 and there is a single output signal. The classical gates can be reversible or non-reversible. The NOT gate is an example of a reversible gate. For example, to see this clearly, consider the truth table representing the action of a XOR gate for two input classical bits, A and B. The output is given by  $A \oplus B$ . We mention this along with other two logic gates in the table below.

A	B	AND ( $A \cdot B$ )	OR ( $A + B$ )	XOR( $A \oplus B$ )
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

This can be extended easily to have three input lines, the output is given by  $A \oplus B \oplus C$  and will be high (1) if the number of highs in the input is an odd number. The fact that there is a single output line clearly implies that we can never run this operation backwards. Given an output, we can never know what was the actual input. So XOR seems like a non-reversible gate. However, if we allow a garbage output (say A), then XOR can be made reversible. The AND gate is not reversible (even if we allow an extra output bit). NAND and NOR are the universal gate set for classical computers (which means that any gates can be constructed from them). The representation of a NAND gate made up of two transistors is shown below.

### 3.1 States on the Bloch sphere and gates

The inputs for the quantum case are known as ‘states’. They can either be  $|0\rangle$ ,  $|1\rangle$  or in superposition of these two states or in some entangled state of these qubits. We will see this in more detail later on. The classical logic gate XOR we considered above, or any other logic gate behaves very differently compared to a quantum logic gate. In quantum circuits, the data (or states) are represented by ‘qubits’, the operations are described by unitary quantum logic gates and the results are obtained by doing measurements. The first point of difference is that in this case, the number of inputs is the same as the number of outputs. The circuit can be run backwards, and we can reproduce input if we want. One striking property of quantum logic gates is that they can change the inputs in ways which cannot be done classically. One example of this is to look at Hadamard



**Figure 3.** The construction of NAND gate using transistor and diodes. The Z is the output and the T shaped sign signals the input  $+V$  voltage. We say the input is high (1) if it is  $+V$  while it is 0 if grounded.

gate (H). If we take an input qubit to be  $|0\rangle$  and then act on it with  $H$  gate, we have:

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

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

which is now in a superposition state. The Hadamard gate can also be expressed as a  $\pi/2$  rotation around the Y-axis, followed by a  $180^\circ$  rotation around the X-axis i.e.  $H = \sqrt{XY}$ . We can represent this gate by the following matrix:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \langle 0| + \frac{|0\rangle - |1\rangle}{\sqrt{2}} \langle 1| \quad (3.1)$$

A natural way of representing the qubit state is by using the Bloch sphere<sup>16</sup> representation as shown in Fig. 4. The Hadamard transformation (also known as ‘Walsh-Hadamard’) is a special case of the more general Fourier transform (FT) which plays a major role in quantum algorithms. One special property of this gate is that  $H = H^{-1}$  and therefore:

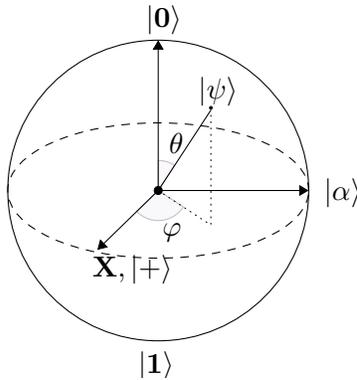
$$H\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right) = |0\rangle.$$

One of the frequently used quantum logic gate is the quantum analog of XOR gate and is known as CNOT (controlled NOT) or CX gate. The matrix representation of CNOT is given by:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (3.2)$$

and the truth table in this case can be written as:

<sup>16</sup>Note that this is mathematically referred to as ‘Riemann sphere’ and is denoted as  $\mathbb{C}\mathbb{P}^1$  which is  $\equiv SU(2)/U(1)$



**Figure 4.** The representation of the possible states of single qubit on the Bloch sphere. We have  $|\alpha\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$  and  $|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\varphi}\sin(\theta/2)|1\rangle$  which is just  $P(\phi)R_y(\theta)|0\rangle$ . The  $Z$ -basis is the ‘computational basis’.

$ A\rangle$	$ B\rangle$	$ A\rangle$	$ A \oplus B\rangle$
$ 0\rangle$	$ 0\rangle$	$ 0\rangle$	$ 0\rangle$
$ 0\rangle$	$ 1\rangle$	$ 0\rangle$	$ 0\rangle$
$ 1\rangle$	$ 0\rangle$	$ 1\rangle$	$ 1\rangle$
$ 1\rangle$	$ 1\rangle$	$ 1\rangle$	$ 0\rangle$

This can also be written as:

$$\text{CNOT} = |0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes \sigma_x, \quad (3.3)$$

where  $\sigma_x$  is one of the Pauli matrices. For this gate, we have two outputs (unlike classical XOR gate) and the first output is sometimes called ‘garbage output’ since it is just the first input. The second input ( $|B\rangle$ ) is called the ‘target’ while the first is called the ‘control’ (or control line) for obvious reasons. The CNOT gate is a two-qubit operation, in which the first qubit is usually referred to as the control qubit and the second qubit as the target qubit. Now suppose an input

state is given by  $|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \\ 0 \\ 0 \end{pmatrix}$ , it is easy to check that if we pass it through CNOT gate, it will

remain unchanged. We leave this simple exercise for the reader. There are many other important quantum gates such as the Pauli-X (simply  $X$ ) which is a rotation through  $\pi$  radians around the  $x$ -axis and similar for other directions ( $y$  and  $z$ ):

$$X = \sigma_x = \sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$Y = \sigma_y = \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

$$Z = \sigma_z = \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

These three matrices also have corresponding rotation matrices, such as  $R_X(\theta) = \exp(-iX\theta/2)$ .  $X$  basis is also called Hadamard basis, since it can be generated from computational basis ( $Z$ ) by acting by  $H$ .

EXAMPLE 2: Show that any single qubit unitary can be written as:

$$U = \begin{bmatrix} e^{i(\alpha-\beta/2-\delta/2)} \cos(\theta) & e^{i(\alpha-\beta/2+\delta/2)} \sin(\theta) \\ e^{i(\alpha+\beta/2-\delta/2)} \sin(\theta) & e^{i(\alpha+\beta/2+\delta/2)} \cos(\theta) \end{bmatrix}. \quad (3.4)$$

Furthermore, show that  $U = e^{i\alpha}R_z(\beta)R_y(\gamma)R_z(\delta)$ .

□ We now give the proof of the above example. We can define general  $U$  as:

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

We have constraints:  $|a|^2 + |c|^2 = 1$ ,  $|b|^2 + |d|^2 = 1$ , and  $a^*b + c^*d = 0$ . Then we have,

$$U = \begin{pmatrix} e^{\alpha_{11}} \cos \theta & e^{\alpha_{12}} \sin \theta \\ e^{\alpha_{21}} \sin \theta & e^{\alpha_{22}} \cos \theta \end{pmatrix}.$$

The last condition (orthogonal columns) gives:  $e^{i(\alpha_{11}-\alpha_{12})} + e^{i(\alpha_{21}-\alpha_{22})} = 0$ , where  $\alpha_{11}$  is related to the other three by  $\alpha_{11} = \alpha_{12} + \alpha_{21} - \alpha_{22} + \pi$ . Now we can set  $\theta = \gamma/2$ ,  $\alpha_{12} = \alpha - \beta/2 + \delta/2 + \pi$ ,  $\alpha_{21} = \alpha + \beta/2 - \delta/2$ , and  $\alpha_{22} = \alpha + \beta/2 + \delta/2$ . □

Another useful quantum gate which is an extension of CNOT gate is known as controlled-controlled NOT (CCNOT) gate or Toffoli gate. This is a three-qubit operation defined by:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \quad (3.5)$$

resulting in

$$\text{---}\boxed{H}\text{---} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad |0\rangle \text{---}\boxed{H}\text{---} |+\rangle$$

$$\text{---}\boxed{X}\text{---} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad |0\rangle \text{---}\boxed{X}\text{---} |1\rangle$$

$$\text{---}\boxed{Z}\text{---} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad |1\rangle \text{---}\boxed{Z}\text{---} -|1\rangle$$

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

$$\text{---}\boxed{P}\text{---} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$$

$$\text{---}\boxed{S}\text{---} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

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

**Figure 5.** Definition of Hadamard ( $H$ ), Pauli ( $X, Y, Z$ ), Phase ( $P$ ),  $\pi/2$  phase ( $S$ ) and  $\pi/8$  ( $T$ ) single qubit gates and example of how some of them act on a single qubit.

$$\begin{aligned} |000\rangle &\rightarrow |000\rangle \\ |001\rangle &\rightarrow |001\rangle \\ |010\rangle &\rightarrow |010\rangle \\ |011\rangle &\rightarrow |011\rangle \\ |100\rangle &\rightarrow |100\rangle \\ |101\rangle &\rightarrow |101\rangle \\ |110\rangle &\rightarrow |111\rangle \\ |111\rangle &\rightarrow |110\rangle \end{aligned} \tag{3.6}$$

► QUESTION 1: Write the  $Z$  and  $P$  gate in the outer product notation and compute the action of  $P$  on a general qubit state.

One of the other important operations is the swapping of qubits, represented as:

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

The gate achieves  $|a, b\rangle \rightarrow |b, a\rangle$  and can be written in terms of CNOT gates as: <sup>17</sup>

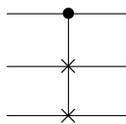
$$\begin{array}{c} |a\rangle \\ |b\rangle \end{array} \begin{array}{c} \times \\ \times \end{array} = \begin{array}{c} \bullet \\ \oplus \end{array} \begin{array}{c} \oplus \\ \bullet \end{array} \begin{array}{c} \bullet \\ \oplus \end{array} \quad (3.7)$$

One of the gates which is often used to reorder or swap qubits is the controlled SWAP (also known as ‘Fredkin gate’) is an important gate that can be implemented as: `qc.cswap(0,1,2)`. We can always see how this gate can be decomposed in terms of CNOT gate as:

`qc.decompose().draw(output='mpl', style='iqx')`. The representation of the gate is given below:

$$\text{CSWAP} = |0\rangle\langle 0| \otimes \mathbb{1} \otimes \mathbb{1} + |1\rangle\langle 1| \otimes \text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (3.8)$$

and is represented by the following symbol:



which means in our notation,  $|0, a, b\rangle \rightarrow |0, a, b\rangle$ , and  $|1, a, b\rangle \rightarrow |1, b, a\rangle$  respectively. The use of control feature can be extended to any unitary gate. For example, consider the controlled  $RX$  gate where the transformation is defined by the matrix:

$$\text{RX} = \begin{pmatrix} \cos(\theta/2) & -i \sin(\theta/2) \\ -i \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}.$$

<sup>17</sup>We can see this as follows:  $|a, b\rangle \rightarrow |a, a \oplus b\rangle$  after application of first CNOT gate. Then the second CNOT (note with control now down and target on upper qubit) transforms  $|a, a \oplus b\rangle \rightarrow |a \oplus (a \oplus b), a \oplus b\rangle = |b, a \oplus b\rangle$ . Then the last CNOT transforms this to the desired  $|b, a\rangle$

If we want to build a controlled-RX gate then the matrix is given by a  $4 \times 4$  matrix:

$$\text{CRX} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta/2) & 0 & -i \sin(\theta/2) \\ 0 & 0 & 1 & 0 \\ 0 & -i \sin(\theta/2) & 0 & \cos(\theta/2) \end{pmatrix}. \quad (3.9)$$

This gate is included in QISKIT like other gates we have discussed above. It can simply be called by doing `qc.crx( $\pi/2, 0, 1$ )` but double controlled RX gate is not included in usual gate library in QISKIT and we leave the design of this gate for the interested reader.

► QUESTION 2: Construct controlled-controlled-RX gate matrix and then implement it in QISKIT. Check that it has the correct behaviour by checking how  $|110\rangle$  and  $|000\rangle$  transform respectively.

► QUESTION 3: Show that the controlled-Hadamard (CH) gate given by:

$$\text{CH} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}, \quad (3.10)$$

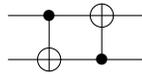
can be written as  $|00\rangle\langle 00| + |01\rangle\langle 01| + \frac{1}{\sqrt{2}}(|10\rangle\langle 10| + |10\rangle\langle 11| + |11\rangle\langle 10| - |11\rangle\langle 11|)$ .

► QUESTION 4: Show that

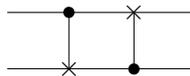
$$H \cdot S \cdot T \cdot H |0\rangle = \frac{1}{2} \left[ (1 + e^{i3\pi/4}) |0\rangle + (1 - e^{i3\pi/4}) |1\rangle \right]$$

where  $H$ ,  $S$ , and  $T$  are the usual gates defined in the text.

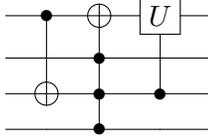
A simple circuit with two CNOT gates is given as:



Though we will use the modern notation for CNOT, the old notation used by Feynman [18] was instead:



Furthermore, a slightly more complicated circuit with multiple control lines and an arbitrary gate  $U$  can be drawn as:



It is easy for the reader to see that if we have two qubits and act with  $H$  on the first qubit and then apply CNOT gate, it results in an entangled state. For three qubits, this is more interesting. Three qubits can be entangled in two ways which are not related to each other. One of them is called the GHZ state and the other is known as W-state [19]. The GHZ state is fully separable, while the W state cannot be separated by any local operations and classical communication (LOCC). These are defined as follows:

$$\text{GHZ}_3 = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle),$$

$$\text{W}_3 = \frac{1}{\sqrt{3}}(|001\rangle + |010\rangle + |100\rangle).$$

It is straightforward to extend  $W$  state to its  $n$ -qubit version as:

$$|W\rangle = \frac{1}{\sqrt{n}}(|10\dots 00\rangle + |01\dots 00\rangle + \dots + |00\dots 01\rangle). \quad (3.11)$$

We can also define something called the ‘generalized’ GHZ state defined as:

$$\text{GHZ}_n = \cos \theta |000\dots 000\rangle + \sin \theta |111\dots 111\rangle. \quad (3.12)$$

It is easy to implement these states in QISKIT and we leave that for the reader. Going beyond the discussion of single gates, for several purposes, some collection of quantum gates is often useful. When these gates are collected together, they are referred to as a ‘set’. For example, one of the example is the stabilizer set, which is given by group of three gates:  $\{H, \text{CNOT}, P\}$  while the Clifford group is generated by three gates:  $\{H, \text{CNOT}, S\}$  gates. These gates are important for error-correcting codes. Note that  $\{\text{CNOT}, \text{single-qubit unitaries } (SU(2))\}$  forms a universal set of gates but it is infinite. We would like this to be close to some finite gate set. A finite gate set can only produce a countable set of gates and one would think that every unitary cannot be obtained by this method. However, we do not need to get every unitary and this is where one of the most important theorems in quantum information/computation comes into play. Solovay-Kitaev (SK) theorem argues that if a single-qubit quantum gates generate a dense subset of  $SU(2)$  then that set is guaranteed to fill  $SU(2)$  quickly (means it is possible to obtain decent approximation to any desired gate using gates from the generating set). We say that a subset  $\mathcal{S}$  of a topological space  $X$  is dense if every point  $x \in X$  either belongs to  $\mathcal{S}$  or is a limit point of  $\mathcal{S}$ . The special unitary group  $SU(N)$  is defined as the set of all  $N \times N$  matrices with unit determinant. For example, the Pauli matrices are the three generators of  $SU(2)$ . There are two universal (approximately) gate sets:  $\{H, \text{CNOT}, \pi/8\}$  as shown in Ref. [20] and  $\{H, \text{CCNOT}, \pi/4\}$  as shown in Ref. [21] and CNOT plus all single-qubit gates [22]. Suppose we consider the gate set  $\{H, \text{CNOT}, T(\pi/8)\}$  and

ask how efficiently a given unitary transformation  $U$  can be implemented. This situation might arise due to the limitation of applying only single-qubit gate most prominently when one required fault-tolerant quantum computation where the features of fault-tolerance are available for selected gates in Clifford group and  $\pi/8$  gate. We would like to have polynomial (in precision and number of qubits) number of gates selected from this set. We ask this question because in general most unitary transformations cannot be approximated precisely. The SK theorem can be useful in this case. A direct consequence of this theorem is that a quantum circuit of  $N$  constant-qubit gates can be approximated to  $\epsilon$  error (in operator norm) by a quantum circuit of  $\mathcal{O}(N \log^c(N/\epsilon))$  gates from a desired finite universal gate set. Different proofs of the theorem give different values for  $c$ , in Ref. [23], it was shown to be  $\sim 3.97$ . This theorem computes how many gates are needed to approximate a given operation for the specified accuracy  $\epsilon$  and is polylogarithmic in  $1/\epsilon$ . We now write down the formal statement as below:

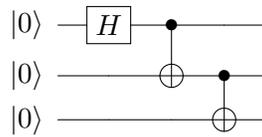
*Solovay-Kitaev theorem:* If  $\mathcal{S}$  is a finite set of 1-qubit gates which is universal and if for any gate  $g \in \mathcal{S}$ , the inverse  $g^{-1}$  can be achieved by finite sequence of gates in  $\mathcal{S}$ , then any 1-qubit gate and hence (any unitary transformation) can be approximated using  $\mathcal{O}(\log^c(1/\epsilon))$  gates with  $c < 4$ . The basic algorithm is given below:

- **function Solovay-Kitaev/SK (UnitaryGate  $U$ , depth  $n$ )**  
 # Read gate to be approximated and accuracy desired ( $\epsilon \rightarrow 0$  as  $n \rightarrow \infty$ )
- **if ( $n = 0$ ), return basic approximation to  $U$**   
 # The function is recursive, so that to obtain an  $n$  approximation to  $U$ , it will call itself to obtain  $n - 1$  -approximations to the unitary.
- **else set  $U_{n-1} = \text{SK}(U, n-1)$  set  $V, W = \text{GC-Decompose}(UU^\dagger)$ ,**  
**set  $V_{n-1} = \text{SK}(V, n-1)$ , set  $W_{n-1} = \text{SK}(W, n-1)$ , return  $U_n = V_{n-1}W_{n-1}V^\dagger W_{n-1}^\dagger$**

A single-gate set of universal quantum gates can also be formulated using the three-qubit Deutsch gate given by:

$$\mathbb{D} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & i \cos(\theta) & \sin(\theta) \\ 0 & 0 & 0 & 0 & 0 & 0 & \sin(\theta) & i \cos(\theta) \end{pmatrix}. \quad (3.13)$$

► **QUESTION 5:** What does the following quantum circuit do? By what name do we know the final state known as?



► QUESTION 6: Check that CNOT results in the following transformations given below, i.e.,  $\text{CNOT}(X \otimes \mathbb{1}) \rightarrow (X \otimes X)\text{CNOT}$  or alternatively  $\text{CNOT}(X \otimes \mathbb{1})\text{CNOT}^\dagger \rightarrow (X \otimes X)$  and so on.

- $X \otimes \mathbb{1} \rightarrow X \otimes X$
- $\mathbb{1} \otimes X \rightarrow \mathbb{1} \otimes X$
- $Z \otimes \mathbb{1} \rightarrow Z \otimes \mathbb{1}$
- $\mathbb{1} \otimes Z \rightarrow Z \otimes Z$

It is also known that this gate can be constructed from a sequence of two-qubit gates [24]. It can also be constructed from a two-qubit gate known as ‘Barenco gate’ [25] given by:

$$\mathbb{B} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\alpha} \cos(\theta) & -ie^{i(\alpha-\phi)} \sin(\theta) \\ 0 & 0 & -ie^{i(\alpha+\phi)} \sin(\theta) & e^{i\alpha} \cos(\theta) \end{pmatrix} \quad (3.14)$$

In fact, it was shown in Ref. [26] that the set  $\{T, H\}$  is universal and was shown to be true by a short proof in Ref. [27]. The fact that  $T$  can perform all classical reversible computation makes it clear that  $H$  gate which is in fact a quantum Fourier transform (QFT) over  $\mathbb{Z}_2$  is the real ‘quantum’ difference!

QUESTION 7: Show that  $H$  can be written for one-qubit as:

$$H = \frac{1}{\sqrt{2}} \left[ (|0\rangle + |1\rangle) \langle 0| + (|0\rangle - |1\rangle) \langle 1| \right].$$

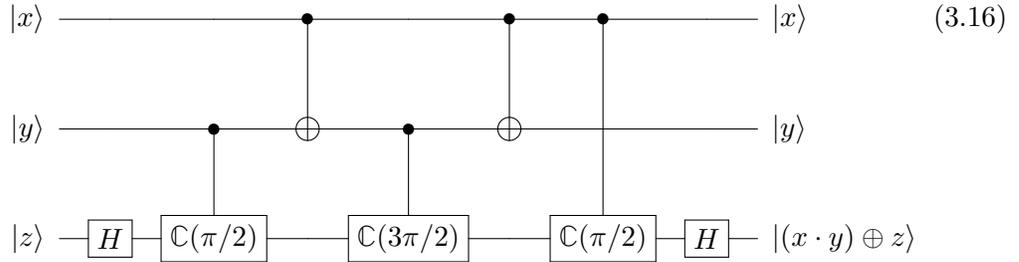
Show that this transform for  $n$ -qubits i.e.,  $H^{\otimes n}$  can be written as:

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

► QUESTION 8: Consider a control line in a superposition state and target in  $|0\rangle$ . Pass this through the CNOT gate and write the result (a Bell state). Show that the circuit given below performs a measurement in the basis of Bell states. This exercise is reproduced from Exercise 4.33 of [11]. Find the measurement operators?



► QUESTION 9: Show that the circuit below is another way of implementing Toffoli gate. Find another decomposition of Toffoli gate?



### 3.2 General quantum operations

Let us consider an orthonormal basis  $|\phi_k\rangle$ , we can write a state as  $|\psi\rangle = \sum_k \alpha_k |\phi_k\rangle$ . von Neumann (vN) measurement of  $|\psi\rangle$  with respect to the  $\phi$  basis is described by orthogonal projectors  $\{|\phi_k\rangle, \langle\phi_k|\}$  and will have the output  $k$  with probability:

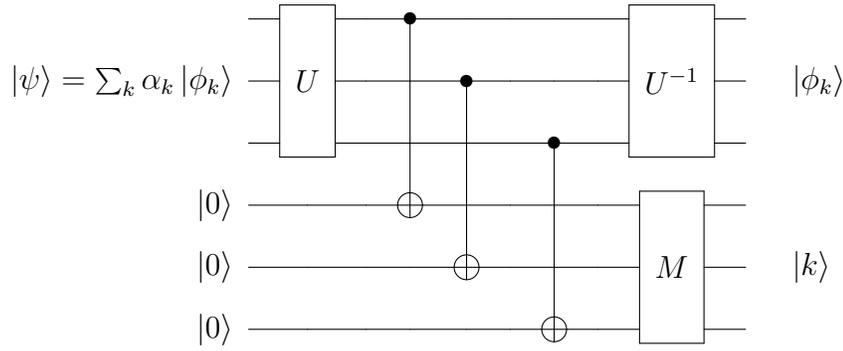
$$\text{Tr}(|\psi\rangle \langle\psi| |\phi_k\rangle \langle\phi_k|) = |\alpha_k|^2.$$

von Neumann (vN) measurements are a special kind of projective measurement (which is also known as Lüders measurement) which is complete. This is often used in quantum computing and communication. We give the circuit which can implement a vN measurement below.

In fact, the idea of projective measurements is related to identifying which state the system is in from the possible set of mutually exclusive states. Suppose, the atom is either in state  $|a\rangle$  or  $|b\rangle$ , through projective measurement we can find out the state it is in. These are based on projection operators  $P$  which are Hermitian operators ( $P = P^\dagger$ ,  $P^2 = P$ ). We refer to two projection operators  $P_1$  and  $P_2$  as being orthogonal if  $P_1 P_2 |\psi\rangle = 0$ . The probability of obtaining output  $k$  (as discussed above) is  $\text{Prob.}(k) = \langle\psi|P_k|\psi\rangle = \text{Tr}(P_k |\psi\rangle \langle\psi|)$ . These measurements lead to the collapse of the wave function and the state after measurement  $\psi'$  is given by:

$$\psi' = \frac{P_k |\psi\rangle}{\sqrt{\langle\psi|P_k|\psi\rangle}}. \quad (3.17)$$

Projective measurements (of which von Neumann is a special case) can be generalized as well but we will not discuss it here and refer the reader to textbooks which discuss quantum measurements in depth.



**Figure 6.** One way of implementing vN measurement through ancillary register. The measurement  $M$  measures  $|k\rangle$  with probability,  $|\alpha_k|^2$  while the map through  $U^{-1}$  gives the state  $|\phi_k\rangle$  in the main register. One can also directly measure instead of introducing ancillary register. Note that  $U$  is a unitary transformation which implements basis change to the computational basis. The use of CNOT gate to copy to the ancillary register might seem illegal but note that we are just doing reversible transformation copying the computational basis states, we are not cloning/copying arbitrary superposition state.

**EXAMPLE 3:** What is the output of the quantum circuit given below? Assume  $U$  is some unitary gate.



□ The initial state is given by:  $|\psi_0\rangle = |0\rangle |\psi_{in}\rangle$ . Once we apply the  $H$  gate it becomes,  $|\psi_1\rangle$  which is given by:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}} \left( |0\rangle |\psi_{in}\rangle + |1\rangle |\psi_{in}\rangle \right)$$

Then, if we act on this with the  $U$  as given in the Figure, we get:

$$|\psi_2\rangle = \frac{1}{\sqrt{2}} \left( |0\rangle |\psi_{in}\rangle + |1\rangle U |\psi_{in}\rangle \right)$$

and now we again act with the  $H$  gate,

$$|\psi_3\rangle = \frac{1}{2} \left( (|0\rangle + |1\rangle) |\psi_{in}\rangle + (|0\rangle - |1\rangle) U |\psi_{in}\rangle \right)$$

and this is equal to:

$$= \frac{1}{2} \left( |0\rangle (\mathbb{1} + U) |\psi_{in}\rangle + |1\rangle (\mathbb{1} - U) |\psi_{in}\rangle \right)$$

□

### 3.3 Entanglement

One of the distinguishing features of quantum mechanics is ‘entanglement’<sup>18</sup>. Though there exists various definitions of entanglement of varying complexity, for our purposes, a simple one will suffice. We say that the state is entangled when it cannot be written as a product state. It is often useful to check whether a given state is separable (not entangled) or not. In order to show whether the state is entangled or not, it is easiest to compute the reduced density matrix and compute  $\text{Tr}\rho_A^2$  and see if it is 1 or not (for normalized states). It is a separable state (not entangled) iff  $\text{Tr}\rho_A^2 = 1$ . Suppose we have a Bell state:  $|\Psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ , then we have:

$$\rho_A = \text{Tr}_B(|\Psi\rangle\langle\Psi|) = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|) = \frac{1}{2}\mathbb{1}$$

$\text{Tr}(\rho_A^2) = \frac{1}{4}\text{Tr}(\mathbb{1} \cdot \mathbb{1}) = \frac{1}{2}$ . Hence, this is entangled. We have used the shorthand notation:  $\text{Tr}_B(\rho_{AB}) = \rho_A$  above.

► QUESTION 10: Compute  $\text{Tr}(\rho_A^2)$  and check if the state given below is separable?

$$|\psi\rangle = \frac{1}{2}(|00\rangle + |10\rangle - |01\rangle - |11\rangle). \quad (3.19)$$

Once you find it is separable, write it in terms of  $|\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$ . Now consider a general state  $|\psi\rangle = a|00\rangle + b|10\rangle + c|01\rangle + d|11\rangle$  and find the condition iff the state is to be a separable one. Now think of a measure using  $a, b, c, d$  of how we can quantify the extent of entanglement. Clearly for this case it should vanish. In later part of these notes, we will redo this exercise using QISKIT, see Question A.2. In the later part of the notes, we will encounter a name for the function which can be built out of  $a, b, c, d$ .

The above arguments can be systematically understood as follows. Consider a density matrix  $\rho$  written as:

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|. \quad (3.20)$$

If the system is in a pure state then all the  $p_i$ , except one, is zero, and we get  $\rho = |\psi\rangle\langle\psi|$ . If the  $\rho$  is a pure state, then the following condition are both sufficient and necessary to prove that it is in such a purity:  $\rho = \rho^\dagger$ ,  $\text{Tr}\rho = \text{Tr}\rho^2$ . Usually one takes the  $\rho$  to be appropriately normalized such that  $\text{Tr}\rho = 1$  which is equivalent to saying that  $\text{Tr}\rho = \text{Tr}\left(\sum_i p_i |\psi_i\rangle\langle\psi_i|\right) = 1$  or  $\text{Tr}\rho = \sum_n \langle n|\rho|n\rangle = \sum_n \langle n|\psi\rangle\langle\psi|n\rangle = 1$  as per taste. It is straightforward to show that  $\text{Tr}\rho = \text{Tr}\rho^2$ . We will just show that  $\rho = \rho^2$ , which can be done by writing:  $\rho^2 = |\psi\rangle\langle\psi|\psi\rangle\langle\psi| = \rho$ , where the sums have been suppressed. Now consider a mixed state, and we will show that  $\text{Tr}(\rho_{\text{mixed}})^2 < \text{Tr}\rho_{\text{mixed}}$ . We also need to show that if  $\rho^2 = \rho$ , the state is pure. We first note that since  $\rho$  is Hermitian, the

<sup>18</sup>This famously prompted Einstein to complain about spooky action at a distance (in German ‘spukhafte Fernwirkungen’)

eigenvalues are real and the corresponding eigenvectors can be made orthonormal, the proof for this proceeds as follows,

$$\rho = \sum_i \lambda_i |\lambda_i\rangle \langle \lambda_i| \quad ; \text{ Spectral decomposition} \quad (3.21)$$

$$= \sum_i \lambda_i^2 |\lambda_i\rangle \langle \lambda_i| \quad ; \quad (\because \rho = \rho^2) \quad (3.22)$$

This implies that the eigenvalues are either 0 or 1. Hence,  $\lambda_i = 1$ , for some  $i = p$  and 0 for  $i \neq p$ . Hence,  $\rho = \sum_i \lambda_i |\lambda_i\rangle \langle \lambda_i| = |\lambda_p\rangle \langle \lambda_p|$  which implies that  $\rho$  is pure.

$$\text{Tr}(\rho_{\text{mixed}})^2 = \sum_i \sum_j p_i p_j |\psi_i\rangle \langle \psi_i| \langle \psi_j| \langle \psi_j| \quad (3.23)$$

$$= \sum_i p_i^2 |\psi_i\rangle \langle \psi_i| < \sum_i p_i |\psi_i\rangle \langle \psi_i| \quad (3.24)$$

$$\neq \rho_{\text{mixed}}. \quad (3.25)$$

For mixed states, the other properties still hold such as,  $\rho = \rho^\dagger$ ,  $\text{Tr}\rho = 1$  and  $\rho \geq 0$  (positivity). So, the measure of violation of  $\text{Tr}\rho^2$  from 1 is a good measure of how mixed the state is. For a maximally mixed state, we have  $\text{Tr}\rho^2 = 1/d$ , where  $d$  is the dimension of the system. This is also called as maximally mixed density matrix. A density matrix corresponding to maximally mixed state also has the largest entropy as:

$$S_{\text{EE}} = -\text{Tr}_A(\rho \ln \rho) \quad (3.26)$$

$$= -d \frac{1}{d} \ln\left(\frac{1}{d}\right) = \ln d. \quad (3.27)$$

Any unitary transformation preserves the pureness of the state intact, i.e., a pure state remains pure.

$$\text{Tr}[(U\rho U^\dagger)^2] = \text{Tr}\left(U\rho \underbrace{U^\dagger U}_1 \rho U^\dagger\right) \quad (3.28)$$

$$= \text{Tr}(U\rho^2 U^\dagger) \quad (3.29)$$

$$= \text{Tr}\rho^2. \quad (3.30)$$

This definition of entropy (called entanglement entropy or von Neumann entropy) for pure states of bipartite system is the most widely used measure to quantify entanglement. As the astute reader might have noticed before, we can also identify if the state is separable by computing the Schmidt number. Consider a state  $|\psi\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$  to be a pure state. Then we have an expansion of the form:

$$|\psi\rangle = \sum_i \lambda_i |a_i\rangle |b_i\rangle,$$

where  $|a_i\rangle$  and  $|b_i\rangle$  are orthonormal states belonging to  $\mathcal{H}_A$  and  $\mathcal{H}_B$  respectively. The  $\lambda_i \geq 0$  are Schmidt coefficients and satisfy  $\lambda_i^2 = 1$ . These are computed by constructing a matrix  $\text{Tr}_B(|\psi\rangle \langle \psi|)$  whose eigenvalues are  $\lambda_i^2$ . The Schmidt number is defined as the number of non-zero  $\lambda_i$ . The state is separable if this is 1 and entangled if it is greater than 1.

### 3.3.1 Partial trace method

In this subsection, we will discuss the idea which facilitates the computation of bipartite entropy. The partial trace,  $\text{Tr}_B$  is a map from the density matrix  $\rho_{AB}$  on some composite system with Hilbert space  $\mathcal{H}_A \otimes \mathcal{H}_B$  onto density matrices  $\rho_A$  on  $\mathcal{H}_A$ . Let us assume that  $\{|\alpha_i\rangle\}$  and  $\{|\beta_i\rangle\}$  are the basis of  $\mathcal{H}_A$  and,  $\mathcal{H}_B$  respectively. Then a density matrix,  $\rho_{AB}$  can be decomposed as:

$$\rho_{AB} = \sum_{ijkl} c_{ijkl} |\alpha_i\rangle\langle\alpha_j| \otimes |\beta_k\rangle\langle\beta_l| \quad (3.31)$$

and the partial trace is given by<sup>19</sup>:

$$\text{Tr}_B \rho_{AB} = \sum_{ijkl} c_{ijkl} |\alpha_i\rangle\langle\alpha_j| \langle\beta_l|\beta_k\rangle \quad (3.32)$$

$$\text{Tr}_B \begin{pmatrix} \rho_{11} & \rho_{12} & \rho_{13} & \rho_{14} \\ \rho_{21} & \rho_{22} & \rho_{23} & \rho_{24} \\ \rho_{31} & \rho_{32} & \rho_{33} & \rho_{34} \\ \rho_{41} & \rho_{42} & \rho_{43} & \rho_{44} \end{pmatrix} = \begin{pmatrix} \rho_{11} + \rho_{22} & \rho_{13} + \rho_{24} \\ \rho_{31} + \rho_{42} & \rho_{33} + \rho_{44} \end{pmatrix} \quad (3.33)$$

while the partial trace over  $A$  is given by:

$$\text{Tr}_A \begin{pmatrix} \rho_{11} & \rho_{12} & \rho_{13} & \rho_{14} \\ \rho_{21} & \rho_{22} & \rho_{23} & \rho_{24} \\ \rho_{31} & \rho_{32} & \rho_{33} & \rho_{34} \\ \rho_{41} & \rho_{42} & \rho_{43} & \rho_{44} \end{pmatrix} = \begin{pmatrix} \rho_{11} + \rho_{33} & \rho_{12} + \rho_{34} \\ \rho_{21} + \rho_{43} & \rho_{22} + \rho_{44} \end{pmatrix} \quad (3.34)$$

A two-qubit state can be expanded in the orthonormal basis  $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$  given by:

$$\rho_{AB} = \rho_{11}|00\rangle\langle 00| + \rho_{12}|00\rangle\langle 01| + \dots + \rho_{44}|11\rangle\langle 11| \quad (3.35)$$

and the partial trace is given by 3.32:

$$\rho_A = (\rho_{11} + \rho_{22})|0\rangle\langle 0| + (\rho_{13} + \rho_{24})|0\rangle\langle 1| + (\rho_{31} + \rho_{42})|1\rangle\langle 0| + (\rho_{33} + \rho_{44})|1\rangle\langle 1| \quad (3.36)$$

One of the other ideas related is - purification. The fact that given mixed density matrix can always be thought of as being obtained from the partial trace of some bigger Hilbert space. This idea is closely related to the phrase – there exists a ‘church of bigger Hilbert space’ supposedly meaning that there always is a place where one can be purified. Suppose  $|i\rangle \in \mathcal{H}_A$  such that:

$$\rho_A = \sum_i \lambda_i |i\rangle\langle i| \quad (3.37)$$

Extend  $\mathcal{H}_A \rightarrow \mathcal{H}_A \otimes \mathcal{H}_B$ , then we can write  $\rho_A$  as,

$$\rho_A = \text{Tr}_B |\Psi\rangle_{AB} \langle \Psi|_{AB} \quad (3.38)$$

which is a pure state where  $|\Psi\rangle_{AB} = \sum_i \sqrt{\lambda_i} |i\rangle_A |i\rangle_B$  Suppose we have a density matrix,  $\rho_{123}$ , and we purify this state by adding a fourth Hilbert space,  $\mathcal{H}_4$ . Then we have,  $S_{1234} = 0$  and from this we get,  $S_{123} = S_4$  and  $S_{234} = S_1$  and  $S_{12} = S_{34}$ .

<sup>19</sup>We note that  $\text{Tr}|\beta_k\rangle\langle\beta_j| = \langle\beta_j|\beta_k\rangle$

► QUESTION 11: Consider the state given by:

$$|\psi\rangle = \cos(\alpha) |00\rangle + \sin(\alpha) |11\rangle, \quad 0 < \alpha < \pi/4 \quad (3.39)$$

Find the eigenvalues of  $\rho$  and  $\rho_2$  and compute the von Neumann entropy.

In addition to von Neumann entropy, there is another notion of entropy often used. This is referred to as ‘relative entropy’. For two density operators  $\rho$  and  $\sigma$ , it is defined as:

$$S_b(\rho||\sigma) = \text{Tr}(\rho \log_b(\rho) - \rho \log_b(\sigma)) \quad (3.40)$$

### 3.3.2 Entangled mixed states

The von Neumann entropy is not a good measure when we deal with entanglement in mixed quantum states. We say a ‘mixed state’ is entangled if it cannot be represented by any mixture of pure states. Unlike the case for pure entangled case, there exists majorly three different definitions of entanglement. These are entanglement of formation (EOF), distillable entanglement (DE), and relative entropy of entanglement (REE)[28, 29]. For example, these definitions have distinguishing features. If we consider EOF of a density matrix, it is zero iff (if and only if)  $\rho$  can be written as a mixture of product states while DE can still be non-zero for this. We will consider only entanglement of formation in this article. The computation of entanglement of formation for a mixed bipartite state ( $\rho_{AB} = \sum_j p_j |\Phi_j\rangle \langle \Phi_j|$ ) is given by:

$$E_f(\rho_{AB}) = \min. \sum_j p_j E(\Phi_j), \quad (3.41)$$

where the minimum is taken over all pure state decomposition of  $\rho_{AB}$ . In addition to EOF, there is another quantitative measure of entanglement known as ‘concurrence’. The entanglement of formation is closely related to another quantity, concurrence (C) through the following relation [30]:

$$E_f(C) = h\left(\frac{1 + \sqrt{1 - C^2}}{2}\right), \quad (3.42)$$

where  $h$  is defined as  $-x \log_2(x) - (1 - x) \log_2(1 - x)$ . In the appendix, we compute this quantity using QISKIT for the interested reader. These computations can also be done using MATHEMATICA and we give a brief sample code using [QI Mathematica package](#) in the Appendix. Alternatively, one can also use the [QuantumFramework](#) package from Wolfram.

► QUESTION 12: Using MATHEMATICA and the package mentioned above, answer the following:

- Pick two random density matrices and compute the relative entropy (as defined in 3.40) between them?

- Consider a density matrix given by:

$$\rho = \begin{pmatrix} 5/12 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 \\ 1/16 & 1/6 & 5/12 \end{pmatrix}. \quad (3.43)$$

Is this a pure or mixed state? Compute the von Neumann entropy.

- Create a random density matrix and compute the von Neumann entropy. Do this several times.

### 3.4 Quantum Fourier Transform (QFT)

The effectiveness of QFT<sup>20</sup> was emphasized in a 1994 paper [31] which eventually was used in Shor's factoring algorithm.

$$\begin{aligned} \text{QFT } |x\rangle &= \frac{1}{\sqrt{2}} \left( \sum_{y=0}^{N-1} e^{2\pi i xy/N} |y\rangle \right) \\ &= \frac{1}{\sqrt{2}} \left( e^{2\pi i x \cdot 0/2} |0\rangle + e^{2\pi i x \cdot 1/2} |1\rangle \right) \\ &= \frac{1}{\sqrt{2}} \left( |0\rangle + e^{i\pi x} |1\rangle \right). \end{aligned}$$

It is easy to check that  $\text{QFT } |0\rangle = |+\rangle$  and  $\text{QFT } |1\rangle = |-\rangle$ . Hence, QFT on a single qubit is just like Hadamard gate. Now let us consider that  $\{0, 1, \dots, N-1\}$  forms an orthonormal basis and let  $|\alpha\rangle = \sum_{j=0}^{N-1} |j\rangle$  denote a state with  $N = 2^n$  for  $n$  qubits. Then QFT transforms the state as:

$$|\alpha\rangle = \sum_{j=0}^{N-1} |j\rangle \rightarrow \sum_{i=0}^{N-1} \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \gamma^{-jk} |k\rangle \quad (3.44)$$

where  $\gamma = e^{-i2\pi/N}$ . Therefore, we have a representation for the QFT as:

$$Q_N = \frac{1}{\sqrt{N}} \sum_{j,k} \gamma^{-jk} |k\rangle \langle j| \quad (3.45)$$

► QUESTION 13: Show that quantum Fourier transform defined by (3.45) is a unitary operation.

<sup>20</sup>We understand that this acronym might be unsuited for field theorists

To see a more general working of QFT, let us assume that  $|x\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \dots \otimes |x_N\rangle$ , then if we perform quantum Fourier transformation on this, we get:

$$\begin{aligned}
\text{QFT } |x\rangle &= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i xy/N} |y\rangle \\
&= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x \sum_{k=1}^n y_k / 2^k} \\
&= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \prod_{k=1}^n e^{\frac{2\pi i x y_k}{2^k}} |y_1 y_2 \dots y_n\rangle \\
&= \frac{1}{\sqrt{N}} \sum_{y_1=0}^1 \sum_{y_2=0}^1 \dots \sum_{y_n=0}^1 \prod_{k=1}^n e^{\frac{2\pi i x y_k}{2^k}} |y_1 y_2 \dots y_n\rangle \\
&= \frac{1}{\sqrt{N}} \left( |0\rangle + e^{\frac{2\pi i x}{2^1}} |1\rangle \right) \otimes \left( |0\rangle + e^{\frac{2\pi i x}{2^2}} |1\rangle \right) \otimes \dots \otimes \left( |0\rangle + e^{\frac{2\pi i x}{2^n}} |1\rangle \right) \quad (3.46)
\end{aligned}$$

We can see that at the qubit level, it maps each of them as:  $|x_k\rangle \rightarrow \frac{1}{\sqrt{2}} \left( e^{\frac{2\pi i x \cdot 0}{2^k}} |0\rangle + e^{\frac{2\pi i x \cdot 1}{2^k}} |1\rangle \right) = \frac{1}{\sqrt{2}} \left( |0\rangle + e^{\frac{2\pi i x}{2^k}} |1\rangle \right)$ . The complexity of QFT is  $\mathcal{O}(n^2)$  where  $n$  is the number of qubits.

### 3.5 Quantum rules and limitations

Since quantum gates implement the principles of quantum mechanics, they are reversible and unitary. The fact that gates are reversible is clear by noting that they have the same number of inputs and outputs. This is not so in classical logic gates like AND, NOR, where there is a single output and the input cannot be created by examining the output. We will now mention several rules and ‘no-go’ theorems which are exclusively quantum in nature. These are important ingredients of the remainder of this article. There are several features of quantum computation that makes it much more interesting than classical counterpart. One of these is the property that arbitrary (unknown) quantum states cannot be cloned. This is known as ‘no-cloning’ theorem [32, 33] It is a simple theorem that has far-reaching consequences. In fact, there is also a reverse theorem known as ‘no-deleting theorem’ [34]. This is also a no-go theorem, which states that given two copies of some arbitrary quantum state, it is impossible to delete one of the copies. It is a time-reversed dual of the no-cloning theorem. These theorems follow from the linearity of quantum mechanics. The no-cloning theorem makes the theory of quantum error correction (discussed later) highly non-trivial since this means that to correct quantum errors due to computation, we cannot simply make backup copies of the quantum state we would like to preserve. Instead, we must protect the original from any likely error for as long as we can.

► **QUESTION 14:** The process of cloning means that there is a unitary transformation  $U$  such that  $U |\psi\rangle |0\rangle \rightarrow |\psi\rangle |\psi\rangle$ . Prove that this is not possible for a general state.

We provide the solution to above question now. Let  $|\psi\rangle$  be  $|0\rangle$  and  $|1\rangle$ , then we have  $U|10\rangle = |11\rangle$  and  $U|00\rangle = |00\rangle$  and by linearity we can write:

$$U\left(\frac{|00\rangle + |10\rangle}{\sqrt{2}}\right) = \frac{1}{\sqrt{2}}\left(|00\rangle + |10\rangle\right),$$

which is equivalent to:

$$U\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)|0\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |10\rangle\right), \quad (3.47)$$

but LHS of (3.47) is equal to by the definition of the  $U$  operator:

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

and hence we have a contradiction, i.e.,

$$\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) \neq \frac{1}{\sqrt{2}}\left(|00\rangle + |10\rangle\right). \quad (3.48)$$

Hence, there cannot be a general  $U$  operator which can clone an arbitrary state. Note that there are some cases where the theorem doesn't apply, which are often a source of confusion. The 'no-cloning' theorem does not apply in two cases. The first is if we have the knowledge of how state was prepared and the second is with reference to classical information. Suppose we have states spanned by orthonormal basis (for example, using CNOT gate, we can do  $|a\rangle \otimes |b\rangle \rightarrow |a\rangle \otimes |a \oplus b\rangle$  and setting  $b = 0$  gives  $|a\rangle \otimes |0\rangle \rightarrow |a\rangle \otimes |b\rangle$ , so we can clone a bit alone.

Now we talk about another theorem which states that it is impossible to distinguish non-orthogonal quantum states. The 'non-distinguishable' theorem can be proved by contradiction as well. Let us suppose that there is a measurement operator  $M$  which has eigenvalues  $m_i$  and has projection operators  $P_i$  of some observable and that it allows us to distinguish between two non-orthogonal states  $|\psi_1\rangle$  and  $|\psi_2\rangle$  with  $\langle\psi_1|\psi_2\rangle \neq 0$ . Then we can write for some state  $|\chi\rangle$  orthogonal to  $|\psi_1\rangle$ :

$$|\psi_2\rangle = \alpha|\psi_1\rangle + \beta|\chi\rangle \quad (3.49)$$

It is easy to show that  $P_2|\psi_1\rangle = 0$  and,

$$1 = \langle\psi_2|P_2|\psi_2\rangle = \langle\psi_2|P_2P_2|\psi_2\rangle = |\beta|^2\langle\chi|P_2|\chi\rangle \leq |\beta|^2. \quad (3.50)$$

The last inequality follows from the completeness property. This implies  $|\beta|^2 = 1$  which means that  $\langle\psi_1|\psi_2\rangle = 0$  which is contradiction. So, it is impossible to unambiguously differentiate between non-orthogonal quantum states.

## Towards Quantum algorithms

---

One major area of present research in quantum computing is building algorithms (quantum) which have substantial speedup<sup>21</sup> compared to its fastest known classical counterpart. This area of research is probably the most interesting and difficult. After close to three to four decades of efforts, we only have a handful of algorithms which demonstrate a clear advantage over classical algorithms. By quantum algorithm, we mean a sequence of unitary steps which manipulate the initial state  $|i\rangle$  such that a measurement of the final state  $|f\rangle$  provides the desired solution. In this section, we will focus on some well-known algorithms and later show how they can be implemented in QISKIT. We will start with probably the simplest of algorithms and one we are well familiar with from an undergraduate course in conventional electronics, i.e., full-adder.

### 4.1 Full-adder

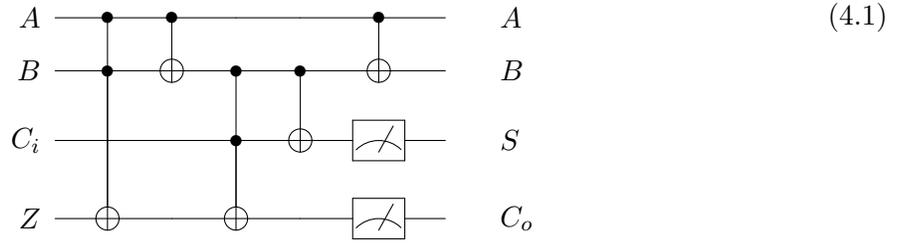
The addition of classical bits using adder circuits is one of the first things one learns in the undergraduate digital electronics course. We will not provide a recap here, but interested reader can refer to the classic textbook by Malvino and Leach [35]. The quantum full-adder is similar to classical full-adder except that now we have an identical number of inputs and outputs because we need quantum circuits to be reversible. Therefore, we define a 4-qubit input, where the input qubits are  $A$ ,  $B$ ,  $C_i$  (Carry in) and null ( $Z$ ). The output qubits are  $A$ ,  $B$ ,  $S$  (Sum) and  $C_o$  (Carry out). The table will be:

$A$	$B$	$C_i$	$Z$	$C_o$	$S$	$A$	$B$
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	0	1	0	1
0	1	1	0	1	0	0	1
1	0	0	0	0	1	1	0
1	0	1	0	1	0	1	0
1	1	0	0	1	0	1	1
1	1	1	0	1	1	1	1

---

<sup>21</sup>In simple terms, the speedup depends on how much quantum mechanics is available.

Once we have prepared the inputs  $A$  and  $B$  it can be fed to the quantum circuit which will implement the full-adder. The circuit is given below:



The implementation of this circuit using QISKIT is given in the Appendix, and we test this for some input from the truth table mentioned above.

## 4.2 Phase kickback

One of the main ingredient for several quantum algorithms is the idea of - ‘phase kickback’. Consider the transformation:  $U|\psi\rangle = e^{i2\pi\phi}|\psi\rangle$ . For the case where  $U$  acts on more than one qubit, we might need more ancillas. We consider two qubits to convey this example. The three parts of this procedure can be summed as follows, 1) An extra qubit which is known as ancilla (control) qubit, 2) Performing Hadamard transform on control qubit(s), and 3) Perform controlled-unitary operation (c-U). It is straightforward to see that the final state will be:

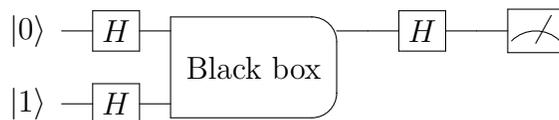
$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{i\phi}|1\rangle\right)|1\rangle, \quad (4.2)$$

where we note that controlled-U acts like  $U|u\rangle = e^{i2\pi\phi}|u\rangle$ . The phase has been kicked back. This is what we set out to do, i.e., getting a phase rotation to our control qubit whereas the phase rotation gate was applied to the bottom qubit. This is different to the common belief that the control bit mostly remains the same and the controlled (target) bit transforms. This is the reason this procedure is called ‘phase kickback’. We can do higher powers of  $U$  to proceed in the same manner. So, if we instead do,  $U^{2^k}$  then we will obtain a phase of  $e^{i2\pi 2^k\phi}$  kicked back. The QISKIT implementation of this algorithm is straightforward, and we do this in the Appendix for the interested reader. Note that sometimes we cannot obtain the exact answer. For example, if we implement the code in the Appendix for  $\theta = 2\pi/3$ , we expect to obtain  $\sim 0.3333$  but indeed get output either to be 0.25 or 0.375. This is part of the algorithm, and it is known that any exact answer will only be obtained with some probability  $p$ . However, we can always increase the number of qubits to get a more precise answer.

## 4.3 Deutsch-Jozsa Algorithm

The first quantum algorithm which showed a speedup over classical algorithm was proposed by Deutsch and later generalized to  $n$ -qubits by Deutsch and Jozsa (DJ) in 1992 [36] and further improved by Cleve et al. [37]. We note that this doesn’t really solve a real problem but is just an example of explicitly showing the quantum advantage. This algorithm makes use of Fourier

transform like several other quantum algorithms such as Simon's algorithm (which we do not discuss in these notes) and several others. All these developments eventually led to Shor's algorithm - the most famous quantum algorithm as of today. We will first explain the Deutsch's algorithm and leave the generalization for the reader to explore from other excellent textbooks and notes available. In simple terms, the statement of the problem is: Suppose we have two boxes and each contains either a red or blue ball inside. We would like to know whether two boxes have the same coloured ball. Mathematically, consider the Boolean function  $f : \{0, 1\} \rightarrow \{0, 1\}$ , then the problem is to find whether  $f(0) = f(1)$  or  $f(0) \neq f(1)$ . We consider the oracle (or black box):  $U_f |x, y\rangle = |x, y \oplus f(x)\rangle$  with  $x, y \in \{0, 1\}$



**Figure 7.** Circuit implementing the Deutsch algorithm.

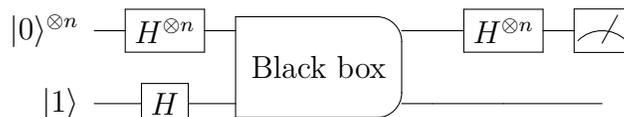
In this case, same color means that function is constant while different result implies it is balanced. So,  $f(0) \oplus f(1) = 0$  if  $f(0) = f(1)$  and 1 otherwise. The sequence of steps as given in the circuit diagram are summarized below:

$$|0, 1\rangle \xrightarrow{H \otimes H} |+, -\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \otimes |-\rangle, \quad (4.3)$$

$$\xrightarrow{U_f} \left[ \frac{1}{\sqrt{2}} (-1)^{f(0)} |0\rangle + \frac{1}{\sqrt{2}} (-1)^{f(1)} |1\rangle \right] \otimes |-\rangle, \quad (4.4)$$

$$\xrightarrow{H \otimes \mathbb{1}} \frac{1}{2} \left[ (-1)^{f(0)} + (-1)^{f(1)} \right] |0, -\rangle + \frac{1}{2} \left[ (-1)^{f(0)} - (-1)^{f(1)} \right] |1, -\rangle. \quad (4.5)$$

Hence, if  $f(0) = f(1)$ , we will measure  $\pm |0, -\rangle$  while if  $f(0) \neq f(1)$  then we will measure  $\pm |1, -\rangle$ . Thus, one call to the oracle (i.e.,  $U_f$  has been used only once above) solves the problem. If we ask as to how many queries to the oracle must be classically made to determine  $f(0) \oplus f(1)$ , it is simple to see that we must make two queries. This corresponds to opening both boxes and checking the colour of the ball. This algorithm is capable of achieving a speedup to any classical algorithm. This becomes drastic with more qubits. This algorithm can be readily extended to  $n$ -qubit and to encourage the reader to generalize this algorithm, we give the circuit diagram as an appetizer below for the DJ algorithm in Fig. 8 below.



**Figure 8.** Circuit implementing the DJ algorithm on  $n$ -qubits.

We might think what makes the single query to oracle possible? The reason is the parallelism achieved due to rules of quantum mechanics, which enables to perform multiple evaluations simultaneously.

► QUESTION 15: Apply the DJ algorithm to input  $|0\rangle|0\rangle|1\rangle$  with  $f(00) = f(01) = 0, f(10) = f(11) = 1$  and write down the output.

#### 4.4 Bernstein-Vazirani algorithm

The Bernstein-Vazirani algorithm is sort of an extension of the Deutsch-Jozsa algorithm discussed above. Its main goal of the algorithm is to show that there can be advantages in using a quantum computer as a computational tool for far more complex problems than those covered by the Deutsch-Jozsa problem. The action of the oracle  $U_f$  is  $|x\rangle \rightarrow (-1)^{f(x)}|x\rangle$  with  $f(x) = x \cdot s$ .

The main steps of the algorithm can be summarized below:

- Suppose the  $n$  qubit state is given by  $|a\rangle$  usually initialized as  $|00000 \dots 000\rangle$ . We first start by applying  $H^{\otimes n}$  i.e.,

$$|a\rangle \xrightarrow{H^{\otimes n}} \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \quad (4.6)$$

- One now applies the oracle to get:

$$\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \mapsto \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle \quad (4.7)$$

- The second application of  $H^{\otimes n}$  then returns:

$$\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)+x \cdot y} |x\rangle = |s\rangle \quad (4.8)$$

We leave the deduction of final step to get  $|s\rangle$  for the interested reader. The similarity between DJ and BV algorithms are: 1) Takes in  $n$  qubits and outputs 0 or 1, 2) Use of  $H$  gate to examine all superposition states. The difference lies in the evaluation and the initial step. DJ determines whether the function is constant or balanced while BV determines the value of function (and search of string,  $s$ ).

► QUESTION 16: Apply the BV algorithm for the three qubit case to find '011' using QISKIT.

#### 4.5 Grover's algorithm

This quantum algorithm was proposed by Grover for structured searching [38, 39]. The goal of this algorithm is to find the solution to the search problem by making calls to the oracle. Suppose we have a list of  $N = 10$  numbers arranged in increasing order from 0 to 9 and the goal is to find 5. On an average, it will take  $\mathcal{O}(N)$  calls to find this. It was shown by that quantum mechanics can help

us solve this problem in  $\mathcal{O}(\sqrt{N})$  attempts. This is a polynomial speedup over the classical case and works by a method called ‘amplitude amplification’. We will see later that Shor’s algorithm offers exponential speedup compared to classical case. This quantum search algorithm consists of repeated application of Grover’s operator  $G$ . There are four steps involved in one iteration (applying  $G$  once). We list them below:

- Apply the oracle  $O$
- Apply the Hadamard transformation on  $n$  qubits denoted by  $H^{\otimes n}$
- Perform phase shift unitary operator where every state in computational basis receives a shift. This operator is given by  $2|0\rangle\langle 0| - \mathbb{1}$
- Apply the Hadamard again, i.e.,  $H^{\otimes n}$

The Hadamard transform (where  $N = 2^n$ ) puts the input state in equal superposition state written as:

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle. \quad (4.9)$$

Using this, we can write the combined effect of all four steps as:

$$G = H^{\otimes n}(2|0\rangle\langle 0| - \mathbb{1})H^{\otimes n}O = (2|\psi\rangle\langle\psi| - \mathbb{1})O = D \cdot O. \quad (4.10)$$

In short, the oracle operation  $O$  reflects the state about  $|\psi\rangle$  and then  $2|\psi\rangle\langle\psi| - \mathbb{1}$  (also called ‘diffuser’  $D$ ) reflects it about  $|\psi\rangle$ . These are the product of two reflections, and the net result is a rotation. One step of Grover’s iteration can be diagrammatically represented by Fig. 9. We also note that  $D = H^{\otimes n}(2|0\rangle\langle 0| - \mathbb{1})H^{\otimes n} = H^{\otimes n}X^{\otimes n}(\text{MCZ})X^{\otimes n}H^{\otimes n}$ . Hence, this can be built from  $H$  gates,  $X$  gates, and single multi-controlled  $Z$  gate. We give the code to implement this algorithm using QISKIT in Sec. A.

#### 4.6 Quantum phase estimation (QPE) algorithm

The QPE algorithm introduced by Kitaev is an important quantum computation algorithm. It plays a central role in several other algorithms. The goal of this algorithm is to estimate  $\theta$  in  $U|\psi\rangle = e^{i2\pi\theta}|\psi\rangle$  for a unitary operator  $U$ . Since the operator is unitary, all eigenvalues have norm of unity. There are several steps involved in QPE which we describe below:

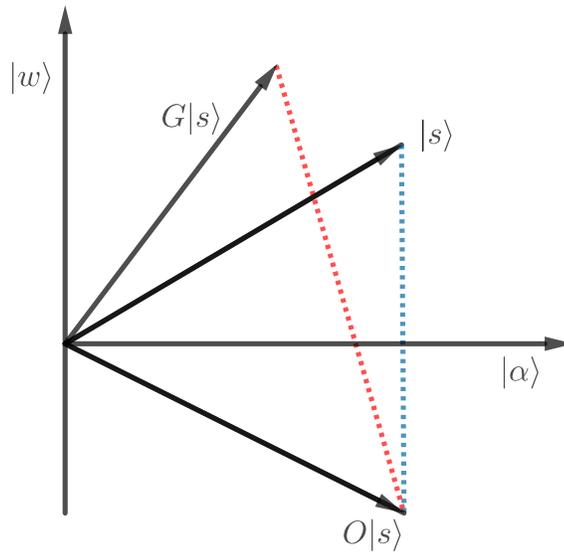
- $|\psi\rangle$  is one set of qubit register (the one below in the figure) and other set is  $n$  qubits counting (first) register. We have  $|\psi_0\rangle = |0\rangle^{\otimes n}|\psi\rangle$
- Apply  $n$ -bit Hadamard gate on the first register to get  $|\psi_1\rangle = \frac{1}{2^{n/2}}(|0\rangle + |1\rangle)^{\otimes n}|\psi\rangle$
- Apply controlled-U (CU) gate (or rather controlled phase with some  $\theta$ )<sup>22</sup> to get

$$|\psi_2\rangle = \frac{1}{2^{n/2}} \left( |0\rangle + e^{2\pi i\theta 2^{n-1}} |1\rangle \right) \otimes \cdots \otimes \left( |0\rangle + e^{2\pi i\theta 2^1} |1\rangle \right) \otimes \left( |0\rangle + e^{2\pi i\theta 2^0} |1\rangle \right) \otimes |\psi\rangle, \quad (4.11)$$

where we note that the rightmost qubit in the first register is  $|q_0\rangle$  (see figure)

---

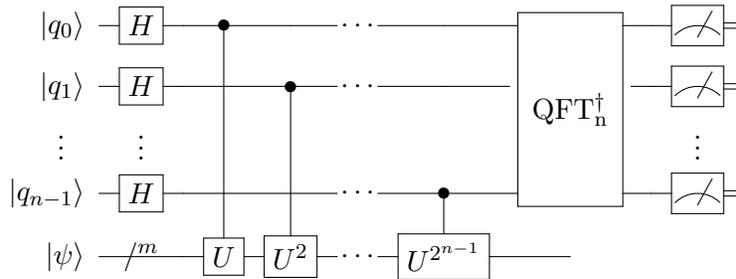
<sup>22</sup>Note that this is what we discussed before in ‘phase kickback’



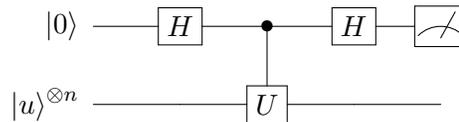
**Figure 9.** Diagrammatic representation of the Grover's algorithm.

- Apply inverse quantum Fourier transform to get

$$|\psi_3\rangle = \frac{1}{2^{\frac{n}{2}}} \sum_{\alpha=0}^{2^n-1} e^{2\pi i \alpha \theta} |\alpha\rangle \otimes |\psi\rangle \xrightarrow{\text{QFT}_n^{-1}} \frac{1}{2^n} \sum_{x=0}^{2^n-1} \sum_{\alpha=0}^{2^n-1} e^{-\frac{2\pi i \alpha}{2^n} (x-2^n \theta)} |x\rangle \otimes |\psi\rangle \quad (4.12)$$



► **QUESTION 17:** Consider the following circuit (this is Problem 5.3 in Ref. [11]):



where  $|u\rangle$  is an eigenstate of  $U$  with eigenvalue equal to  $e^{i\theta}$ . Show that the top qubit is measured to be 0 with probability equal to  $\cos^2(\theta/2)$

#### 4.7 Shor's factoring algorithm

One of the central goals of quantum computation is to identify problem where the quantum offers significant advantage over classical methods. In this regard, the pioneering work of Ref. [40]

showed that it is possible to factor a number using a quantum computer in a polynomial time, unlike the requirement of exponential time with classical computers. This problem is much more interesting than it appears to be at first glance. It is a known fact that multiplying two numbers (say 13 and 73) is much easier than finding prime factors of 949 as it would take more time. This problem is very hard (and sometimes intractable) with large numbers. This was one of the first examples which explicitly showed that quantum algorithms can do exponentially better than classical ones. The factoring algorithm is the flagship result in quantum algorithms of the last three decades. The algorithm efficiently factors a given integer into two integers. For example, we can factor 1,624,637,792,837 into the primes 15,485,867 and 104,911 respectively faster than any classical algorithm. This is a very practical application because in today's world many modern cryptography systems, such as RSA protocol heavily rely upon the fact that factoring such large integers is computationally difficult for any given computer in reasonable time. However, we are far from practical usage of this algorithm since one would ideally need several thousand of qubits to perform such a task and is *well*-beyond the NISQ era. In 2001, it was shown how to successfully factor 15 via Shor's algorithm on a 7-qubit nuclear magnetic resonance (NMR) quantum computer [41]. In fact, few years ago, it was argued in Ref. [42] that it would take about 8 hours to factor 2048-bit RSA integers using 20 million noisy qubits! We can summarize the algorithm in following steps:

1. If  $N$  is even, return the factor 2 (of course this most likely won't happen!)
2. Check whether  $N = a^b$  for integers  $a, b$ . Return  $a$  as a factor if it is true.
3. Pick a random integer  $p$  between 2 and  $N$ . If  $\text{gcd}(N, p) \neq 1$ , we are done and can return the factor. If not, we continue to Step 4 (likely will always happen!).
4. Use an algorithm to find period  $r$  such that  $a^r \bmod N = 1$
5. If  $r$  is even and  $a^{r/2} \bmod N \neq -1$ , then evaluate  $\text{gcd}(a^{r/2} \pm 1, N)$ . If we find non-trivial factor, return that value as a factor otherwise we return to Step 3.

Note that Euclid's algorithm can be used to find  $\text{gcd}(a, b)$  efficiently.<sup>23</sup> Let us clarify the step of period finding for the reader. Consider:  $a^r = 1 \pmod{N}$  where  $r$  is the smallest positive integer. Now suppose we have,  $N = 15, a = 7$  then  $r = 4$ . This is well-defined only if  $N$  and  $a$  are co-prime or relatively prime (i.e., they have no common factor except 1). In MATHEMATICA, this can be evaluated by doing: `MultiplicativeOrder[7, 15]`. The fact that finding period can actually help us find prime factorization can be obtained from lemmas which follow from Fermat's little theorem (1640) i.e., for every  $a$  and prime  $p$ , we have  $a^p = a \pmod{p}$ . An alternate statement of this theorem is also often mentioned:  $a^{p-1} = 1 \pmod{p}$  where  $p$  is prime and  $a$  is any integer not divisible by  $p$ .

---

<sup>23</sup>In his book, 'The Art of Computer Programming, Vol. 2: Seminumerical Algorithms', Knuth comments that this algorithm is the granddaddy of all algorithms, because it is the oldest nontrivial algorithm that has survived to the present day. It was written down by Euclid circa 300 B.C.

## Variational Quantum Eigensolver (VQE)

---

One of the main motivations why quantum computing is required is the necessity to simulate complicated quantum many-body systems or solving large-scale linear algebra problems. These are very challenging (and sometimes not possible) for classical computers due to the high computational cost involved. Even though quantum computers offer a promise to the solution, the error correction and feasibility is still not well understood. One of the major areas of research with respect to how algorithms can be developed is Variational Quantum Algorithms (VQAs). We refer the interested reader to Ref. [43] to start the reference trail. This makes use of classical optimizer and along with quantum circuits has been argued to have a role in wide-ranging areas where quantum computation might be useful in the future. But, in particular, of the earliest and well-known of these VQAs is an algorithm that has drawn considerable interest in the last decade. This is known as VQE and makes use of the well-known variational principle to compute the ground state energy of a given Hamiltonian, which is important for many problems in quantum chemistry, condensed matter physics, and quantum many-body systems in general. One of the main advantages of this method is that it can be used to deal with complex wave functions of wide class of systems in polynomial time when classical methods grow exponentially. Though, this method has not yet truly outclassed the classical algorithms, it has the incredible potential once we enter beyond NISQ era. This algorithm has well-known drawbacks since this is a non-linear optimization problem. We refer the interested reader to Ref. [44] to start the reference trail. The Variational Quantum Eigensolver (VQE) was originally developed by Peruzzo et al. [45] and is among the most promising examples of NISQ algorithms. It aims to compute an upper bound for the ground state energy of some given Hamiltonian. VQE is a mixture of classical and quantum algorithms which can be implemented on NISQ devices and used to obtain the ground state of quantum many-body of given Hamiltonian  $\mathcal{H}$  using some ansatz. It is based on the variational method of quantum mechanics and divides the task of finding the ground state into parts performed by classical computers and quantum computers separately. The basic idea of the algorithm is as follows:

1. Prepare an initial quantum state  $|0\rangle$  on a quantum computer.
2. Generate a good ansatz quantum state  $|\Psi(\Theta)\rangle$  by applying a suitable unitary transformation (quantum circuit composed of quantum gates) as  $|\Psi(\Theta)\rangle = \hat{U}(\Theta) |0\rangle$
3. The energy expectation value is measured on quantum computer as  $E_{\Theta} = \langle \Psi(\Theta) | \mathcal{H} | \Psi(\Theta) \rangle$
4. Update the parameters  $\Theta$  to get smaller energy  $E(\Theta)$  by using a classical optimizing algorithm.
5. Repeat steps 2-5 until desired convergence is attained.

Within VQE, the cost function is defined as the expectation value of the Hamiltonian computed in the trial state. The ground state of the target Hamiltonian is obtained by performing an iterative

minimization of the cost function. The optimization is carried out by a classical optimizer, which leverages a quantum computer to evaluate the cost function and calculate its gradient at each optimization step.

The choice of ansatz is usually determined by hardware availability and efficiency of considering qubits as logical qubits.<sup>24</sup> We refer the interested reader to Ref. [46] for elaborate discussion on hardware-efficient trial states. The VQE that accompanies QISKIT comes with some given choices such as RX, RYZ,  $SU(2)$  etc. with some user defined number of repetitions. To measure the energy expectation value in the third step above, we must have a qubit Hamiltonian of size  $2^n \times 2^n$ . This is usually not an easy task, however this can be simplified considerably by writing the original Hamiltonian in terms of the sum of Pauli operators (X,Y,Z,  $\mathbb{1}$ ) and their tensor products. This can also be understood as the first step even before doing any computation and has to be done only once. The choice of a right optimization method is crucial for the problem. We can choose from a range of methods such as BFGS (Broyden–Fletcher–Goldfarb–Shanno) optimizer method, COBYLA (Constrained Optimization By Linear Approximation optimizer), SLSQP (Sequential Least Squares Programming optimizer), Nelder-Mead or any other depending on the requirement and efficiency required. For example, another example of local optimizer in QISKIT is Powell method, which performs unconstrained optimization; it ignores bounds or constraints and is a conjugate direction method. More details can be found in Ref.[47]. Another interesting method to construct the ground state of an interacting Hamiltonian is the adiabatic state preparation (ASP) method. It works reasonably well when there is no level crossing (i.e., the excited state of the Hamiltonian do not cross each other and ground state). Recently, there have been some interesting work combining the ideas of VQE approach and ASP approach [48].

When we deal with physical systems, we can often express a given Hamiltonian as the linear combination of the tensor product of Pauli matrices  $\sigma_x$ ,  $\sigma_y$ , and  $\sigma_z$  which we call ‘Pauli strings’ i.e.,  $\hat{P} = \otimes_{i=1}^n \hat{\Sigma}$  where  $\hat{\Sigma} = \{\sigma_x, \sigma_y, \sigma_z, \mathbb{1}\}$ . Note that as the Pauli strings form a complete basis, any Hamiltonian of size  $2^N \times 2^N$  can be decomposed into a linear combination of Pauli strings. For example, let us consider the Hamiltonian of some system to be given by:

$$H = \begin{pmatrix} \frac{1}{2} & 0 & -\frac{i}{2} & 0 & 0 & 0 & 0 & -\frac{1}{2} - \frac{i}{2} \\ 0 & \frac{1}{2} & 0 & \frac{i}{2} & 0 & 0 & -\frac{1}{2} + \frac{i}{2} & 0 \\ \frac{i}{2} & 0 & \frac{1}{2} & 0 & 0 & -\frac{1}{2} + \frac{i}{2} & 0 & 0 \\ 0 & -\frac{i}{2} & 0 & \frac{1}{2} & -\frac{1}{2} - \frac{i}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} + \frac{i}{2} & \frac{1}{2} & 0 & \frac{i}{2} & 0 \\ 0 & 0 & -\frac{1}{2} - \frac{i}{2} & 0 & 0 & \frac{1}{2} & 0 & -\frac{i}{2} \\ 0 & -\frac{1}{2} - \frac{i}{2} & 0 & 0 & -\frac{i}{2} & 0 & \frac{1}{2} & 0 \\ -\frac{1}{2} + \frac{i}{2} & 0 & 0 & 0 & 0 & \frac{i}{2} & 0 & \frac{1}{2} \end{pmatrix}$$

which can be written in terms of Pauli operators as:  $H = \frac{1}{2}(\sigma_z \otimes \sigma_y \otimes \sigma_z - \sigma_x \otimes \sigma_x \otimes \sigma_x - \sigma_y \otimes$

---

<sup>24</sup>Usually when we have qubits, they are prone to errors and can make computation incorrect. We want qubits which are ideal or at least not prone to much errors, they are known as ‘logical qubits’. The problem is that usually for every logical qubit we need multiple physical qubits which can sometimes be hundreds of them.

$\sigma_y \otimes \sigma_y + \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1}$ ). The program to do this transformation can be found in the subroutine of the VQE program in Sec. A.

## 5.1 Quantum Anharmonic Oscillator

We will now discuss how to apply VQE to solve (i.e., find the ground state energy) quantum anharmonic oscillator. This example provides sufficient exposure to the basis working of VQE. This problem was also studied in [49] and we will follow their discussion closely. For the ansatz, we choose `EfficientSU2` circuit from QISKIT circuit library, which is a hardware efficient, heuristic ansatz with alternating rotation and entanglement layers. Note that it is possible to calculate excited states of a Hamiltonian based on a technique known as variational quantum deflation (VQD) proposed recently in Ref. [50]. Let us write the Hamiltonian as:

$$\hat{H}' = \hat{H} - g\hat{X}^3 - h\hat{X}^4. \quad (5.1)$$

For the cubic oscillator, if we consider three qubits,  $\hat{H}'$  is a  $8 \times 8$  matrix which for  $g = 0.02$  is given by:

$$\begin{aligned} \hat{H}' = & 4\mathbb{1}_8 - 0.152956\mathbb{1}_4X - 0.5\mathbb{1}_4Z - 0.12289 * \mathbb{1}XX - 0.0629948\mathbb{1}YY - \mathbb{1}Z\mathbb{1} + 0.0237627\mathbb{1}ZX - 0.0280252X\mathbb{1}X \\ & - 0.0561195XXX + 0.0287333XYY + 0.0107047XZX - 0.0280252Y\mathbb{1}Y - 0.0287333YXY - 0.0561195YYX \\ & + 0.0107047YZY - 2Z\mathbb{1}_4 + 0.0872346Z\mathbb{1}X + 0.0842295ZXX + 0.041655ZYY + 0.0207442ZZX \end{aligned} \quad (5.2)$$

The energy levels for cubic anharmonic oscillator ( $h = 0$ ) which we take as example is known to be:

$$E_n = -\frac{g^2}{8} (30n^2 + 30n + 11) + \mathcal{O}(g^4), \quad (5.3)$$

and the ground state energy level upto  $\mathcal{O}(g^6)$  is given by (see Table (3.1) of Ref. [51])

$$E_0 = \frac{1}{2} - \frac{11}{8}g^2 - \frac{465}{32}g^4 - \frac{39709}{128}g^6 \quad (5.4)$$

We provide the code in QISKIT in the Appendix for the interested reader though admittedly it can definitely be improved and optimized of which we have not taken any care except checking that it works. We need to create  $H$  matrix for this system which can be done in MATHEMATICA as below<sup>25</sup> which will be then converted to Pauli string and input to VQE to find  $E_0$ :

```

1 SetDirectory[NotebookDirectory[]];
2 nbits = 3; (* number of qubits *)
3 n = 2^nbits;
4 lattice = Table[(2 a - n - 1)/2, {a, n}]; (* Lattice for position operator *)
5 X = DiagonalMatrix[Sqrt[(2 \[Pi]/n)] lattice]; (* Position operator *)
6 F = (1/Sqrt[n]) Table[NExp[(-I 2 Pi lattice[[j]] lattice[[k]])/n], {j, n}, {k, n}]; (*
   Discrete Fourier transform *)
7 P = Chop[F\[ConjugateTranspose] . X . F]; (* Momentum operator *)
8 A = Sqrt[0.5] (X + I P); (* Annihilation operator *)
9 A1 = Table[If[(j - i) == 1, i^0.5, 0], {i, n}, {j, n}]; (* Annihilation operator in energy

```

<sup>25</sup>Note that there are other ways to do this, but this is probably the cleanest

```

(E) basis *)
10 X1 = Sqrt[0.5] (A1\[ConjugateTranspose] + A1); (* Position operator in E basis *)
11 P1 = I Sqrt[0.5] (A1\[ConjugateTranspose] - A1); (* Momentum operator in E basis *)
12 g = 0.02;
13 h = 0.04;
14 HO[X_, P_] := .5 P . P + .5 X . X; (* Harmonic oscillator Hamiltonian *)
15 H1[X_, P_] := .5 P . P + .5 X . X - g MatrixPower[X, 3]; (* cubic anharmonic Hamiltonian *)
16 H2[X_, P_] := .5 P . P + .5 X . X + h MatrixPower[X, 4]; (* quartic anharmonic Hamiltonian *)
17 HOA[A_] := A\[ConjugateTranspose] . A + .5 IdentityMatrix[n]; (* Harmonic oscillator
Hamiltonian in E basis *)
18 H1A[A_, X_] := HOA[A] - g MatrixPower[X, 3]; (* Cubic anharmonic Hamiltonian in E basis *)
19 H2A[A_, X_] := HOA[A] + h MatrixPower[X, 4]; (* Quartic anharmonic Hamiltonian in E basis *)
20 H = H1A[A1, X1]; (* Make Hamiltonian to export *)
21 hamName = "HO"; (* Set Hamiltonian name for file *)
22 Export["ham_" <> hamName <> ".txt", H, "Table"]; (* Export Hamiltonian to file which would be
read by our QISKIT program! *)
23

```

In fact, an astute reader might ask: What is the point of doing this VQE computation when I can easily find the ground state energy of an  $8 \times 8$  Hamiltonian in three lines by doing:

```

1 from numpy import linalg as LA
2 w, v = LA.eig(H)
3 print ("Ground state energy", w[0])
4

```

To address this, we note that the interest in VQE algorithms is because for a Hamiltonian of size  $2^n \times 2^n$  where  $n$  is a large number and  $H$  is complicated enough that it is not very sparse like it is in this case, then it will take classical computing/exact diagonalization an exponentially longer time and large amount of memory. This is often given the name ‘solution does not scale with system size’. However, if it is assisted through VQE (creating quantum state ansatz and optimizing classically) then the growth with  $n$  and sparseness of  $H$  will be much efficient (at most polynomial!). It must be noted that VQE has already been able to substantially get close to some gold standard in quantum chemistry, however the discussion of this lies beyond the scope of this article. We just point out one reference of the computation done in quantum chemistry that was for quantum simulation of the deuteron binding energy on quantum processors such as IBM QX5 (with 16 SC qubits) and Rigetti 19Q (19 SC qubits) [52] which can be used to start a reference trail.

## 5.2 $O(3)$ non-linear sigma model

We discuss another toy model (familiar to most physicists), the ill-named and famous  $O(3)$  non-linear sigma model<sup>26</sup> in 1+1-dimensions. This was studied using VQE recently in the Ref. [53] on about 10 lattice sites in the weak and strong coupling limit. In this subsection, we will compute the

<sup>26</sup>We thank Kostas Orginos and Felix Ringer for discussions on this subsection

ground state energy for four sites in the strong and weak coupling limits using exact diagonalization. The Hamiltonian of the model is given by:

$$\hat{H} = \frac{1}{2\beta} \sum_i \mathbf{L}_i^2 - \beta \sum_{\langle i,j \rangle} \mathbf{n}_i \cdot \mathbf{n}_j, \quad (5.5)$$

where  $i$  and  $j$  are the nearest neighbour sites on a uni-directional lattice,  $\mathbf{n}$  is a unit 3-vector at site  $i$ ,  $\mathbf{L}$  is the angular momentum and  $\beta$  is the coupling. We consider a four-site lattice for  $\beta = 1/10, 10$  respectively at fixed truncation of  $l_{\max.} = 1/2^{27}$  at each site. The dimension of the local on-site Hilbert space is given by  $\sum_0^{l_{\max.}} (2l + 1)$  which is  $d = 2$  with  $l_{\max.} = 1/2$ . Therefore,  $H$  will be a  $16 \times 16$  matrix. Though one can apply VQE methods on this, as mentioned in the previous subsection, this is not really needed for the simple example we consider here. We can exactly diagonalize (ED) the Hamiltonian by expressing (5.5) in terms of Pauli matrices and compare to the results from Ref. [53]. We find that for  $\beta = 1/10$ , we get  $E_0/N = 3.72778$  while for  $\beta = 10$ , we get  $E_0/N = -2.18$ . The result at large  $\beta$  is inaccurate (not close to convergence) since the truncation effects are important. One obtains  $E_0/N = -5.78$  if  $l_{\max.} = 3/2$  is used for the same lattice. The code for implementing this with  $l_{\max.} = 1/2$  is given below.

```

1 X = {{0, 1}, {1, 0}};
2 Y = {{0, -I}, {I, 0}};
3 Z = {{1, 0}, {0, -1}};
4 Nplus = -1/(3 Sqrt[2]) (X + I Y) ;
5 Nminus = -1/(3 Sqrt[2]) (X - I Y) ;
6 SITES = 4;
7 HH[\[Beta]_, N_] := N (3/(8 \[Beta])) IdentityMatrix[16] + \[Beta] (KroneckerProduct[Nplus ,
    Nminus, IdentityMatrix[2], IdentityMatrix[2]] + KroneckerProduct[Nminus, Nplus,
    IdentityMatrix[2], IdentityMatrix[2]] + KroneckerProduct[Z , Z, IdentityMatrix[2],
    IdentityMatrix[2]]/9) + \[Beta] (KroneckerProduct[IdentityMatrix[2], Nplus , Nminus,
    IdentityMatrix[2]] + KroneckerProduct[IdentityMatrix[2], Nminus, Nplus,
    IdentityMatrix[2]] + KroneckerProduct[IdentityMatrix[2] , Z, Z, IdentityMatrix[2]]/9) +
    \[Beta] (KroneckerProduct[IdentityMatrix[2], IdentityMatrix[2], Nplus , Nminus ] +
    KroneckerProduct[IdentityMatrix[2], IdentityMatrix[2], Nminus, Nplus] +
    KroneckerProduct[IdentityMatrix[2] , IdentityMatrix[2], Z, Z ]/9) + \[Beta]
    (KroneckerProduct[Nminus, IdentityMatrix[2], IdentityMatrix[2], Nplus ] +
    KroneckerProduct[Nplus, IdentityMatrix[2], IdentityMatrix[2], Nminus] +
    KroneckerProduct[Z, IdentityMatrix[2] , IdentityMatrix[2], Z ]/9);
8 HH[0.1, SITES]/SITES // Eigenvalues // Min // N
9 (* Exercise: Write a general $N$ site program. Try ED for 10 sites and compare.*)
10

```

Note that going beyond the  $l_{\max.} = 1/2$  is non-trivial. The size of  $H$  with  $l_{\max.} = 3/2$  and four sites will be  $6^4 \times 6^4$ , and with  $l_{\max.} = 5/2$ , it will be  $12^4 \times 12^4$ . In addition to the  $O(3)$  model,

<sup>27</sup>Note that this is actually with  $q = 1/2$  which corresponds to  $O(3)$  model with  $\theta = \pi$ . If  $q = 0$ , the  $l$  takes integer values starting at 0. The eigenbasis is denoted by  $|qlm\rangle$  rather than the usual  $|lm\rangle$ .

the hybrid quantum algorithms have also been applied to other models such as two-flavor Gross-Neveu model [54]. It will also be interesting to understand these models from the perspective of continuous variable (bosonic) quantum computing methods and some work along these lines have already been done using Xanadu’s simulator [55] where the ground state energy for scalar field theory was computed using quantum imaginary time evolution (QITE) algorithm proposed with and without use of ansatz in Refs. [56, 57].

SECTION 6

## Quantum Error Correction (QEC)

---

Quantum computers often offer drastic speedup to solutions of certain types of problems by using principles of quantum mechanics such as: superposition and entanglement. However, the use of qubits in place of classical bits also makes the quantum computer susceptible to errors unlike anything we know in classical computers. The origin of these errors are primarily due to decoherence - a process in which the environment interacts with the qubits and changes the quantum states, which eventually leads to corrupted results. For example, failure rate is about one error in  $10^{17}$  operations for classical computers, which for quantum computers is much severe. This partly makes the theory of quantum error correction more subtle than classical counterpart, in addition to several reasons below:

- The act of measurement or observation in quantum mechanics destroys the quantum state and makes its recovery impossible. For example, if we have a superposition state, and we perform measurement, we lose the superposition by choosing one outcome.
- No-cloning theorem prohibits copying of any arbitrary quantum state, and this means that repetition code obtained by just duplicating the bit will not work.
- Unlike classical bit, qubit is continuous valued and hence we need infinite precision to locate the error exactly in order to correct it.

The goal of the quantum error correcting (QEC) code is to protect some subspace of the entire Hilbert space  $\mathcal{H}$  from some errors. More precisely, suppose we want to preserve a  $k$ -dimensional subspace (coding space) against some known errors. This is achieved by mapping the states into a larger,  $n$ -dimensional Hilbert space. In that case, we can refer to it as the  $(n, k)$ -quantum code. Sometimes, an alternate definition is also used, where by  $[n, k, d]$  we denote a quantum error-correcting code that uses  $n$  qubits to encode  $k$  qubits with distance  $d$ . There is a well-known Knill-Laflamme theorem [58] for the conditions of quantum error correction. It is stated as follows: Let  $\mathcal{C}$  be a quantum-error correcting code defined as a subspace of the  $n$ -dimensional Hilbert space,  $\mathcal{H}_2^{\otimes n}$ , and let  $\mathcal{E} \subset \mathcal{E}^{2^n \times 2^n}$  be a set of errors. Then  $\mathcal{C}$  can correct  $\mathcal{E}$  if and only if

$$\langle \psi_i | E_a^\dagger E_b | \psi_j \rangle = c_{ab} \delta_{ij},$$

where  $|\psi_i\rangle$  is a base of the subspace that defines the code  $\mathcal{C} \subset \mathcal{H}_2^{\otimes n}$  and  $E_a, E_b \in \mathcal{E}$ . Equivalently, we can also express the above statement as: Let  $C$  be a quantum code and let  $P$  be a projector onto  $C$ . Suppose a list of operation elements are represented by  $\{E_i\}$ . Then, a necessary and sufficient condition for the error-correcting scheme correcting error on  $C$  is:

$$PE_i^\dagger E_j P = \alpha_{ij} P, \quad (6.1)$$

for some Hermitian matrix  $\alpha_{ij}$ . A quantum error-correcting code is a pair  $(C, R)$  consisting of a quantum code and a recovery operator. We will see one example (the Shor's nine-qubit code) for error correction toward the end of this section. In order to implement QEC, one would like to know which gate set are sufficient. The answer to this is given by the Rains-Solovay theorem.<sup>28</sup> It states that: The Clifford group (denoted below by  $C$ ) together with any gate not in this group is universal for quantum computation. However, there are some combination which are most frequently used and proven to be universal. Such examples are  $\{C + \text{CCNOT}\}$ ,  $\{C + \pi/8\}$ , and  $\{C + \text{controlled} - \pi/4\}$ .

In 1995, Shor [59] discussed a way to reduce the rate of decoherence in quantum memory and hence paved a way for practical usage of quantum computers, This work led to lot of interesting developments in the field of 'quantum error correction'. This was a remarkable result since it implied that a quantum state even though it cannot be cloned due to 'no-cloning' theorem can be protected in a noisy environment. One would at this point think - why do we need to correct errors at all? The error correction in quantum computers is more delicate than classical errors. This is partly because error correction is needed to protect the quantum superposition and entanglement, both of which are crucial ingredients of any quantum computation. In the original work, Shor proposed to store the quantum information not in a single qubit, but it was later shown that it can be reduced to five, which is the minimum required [60].

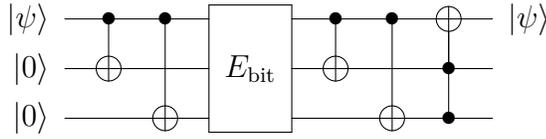
## 6.1 Bit-flip error

Bit flip is defined when we have  $|0\rangle \rightarrow |1\rangle$  or vice versa. Suppose we are interested in sending out a single qubit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  through during which a flip i.e.,  $X|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$  can occur with some probability  $p$ . Such a process is referred to as 'bit-flip'. One way to correct this is to do repetition as follows of the classical bit  $|0\rangle_L \rightarrow |000\rangle$ , where we mean logical qubit by subscript  $L$ . Assuming that the probability of bit-flip as  $p$ , we can show that the total probability of error persisting (i.e., remaining uncorrected or wrongly corrected) is  $3p^2(1-p) + p^3 = 3p^2 - 2p^3$ .

► QUESTION 18: Compute the minimum fidelity attained by the three qubit bit flip?

The circuit to correct bit-flip is given by: For the example, we will take  $E_{\text{bit}}$  to be a  $X$  gate for

<sup>28</sup>This theorem is not often referred to by this name, we call it this following the discussion at: <https://cstheory.stackexchange.com/questions/11308/universal-sets-of-gates-for-su3>



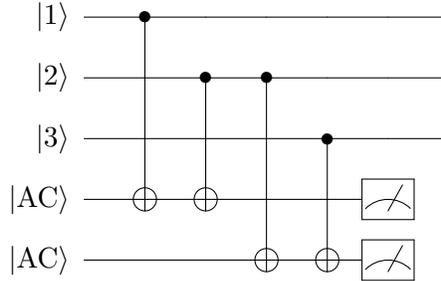
**Figure 10.** Bit flip circuit

now. We have  $Z = |0\rangle\langle 0| - |1\rangle\langle 1|$

$$Z_1 Z_2 \mathbb{1} = (|00\rangle\langle 00| + |11\rangle\langle 11|) \otimes \mathbb{1} - (|01\rangle\langle 01| + |10\rangle\langle 10|) \otimes \mathbb{1} \quad (6.2)$$

If both first and second qubit are same, we get +1 or -1 if they are different. Similarly, we can compute  $Z_2 Z_3$ . Then assuming that there is a single bit flip, we can deduce the following table.

$Z_1 Z_2$	$Z_2 Z_3$	Which bit flipped?
+1	+1	None ( $\mathbb{1}$ )
+1	-1	Third ( $X_3$ )
-1	+1	First ( $X_1$ )
-1	-1	Second ( $X_2$ )



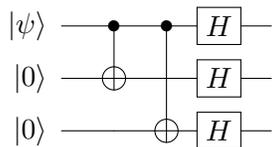
**Figure 11.** A circuit to identify the bit flip with two ancillary bits. This is the error-detection part of the bit flip code. This is followed by recovery procedure.

This step detected the error. Now, in the recovery step, one simply applies  $X$  gate on the flipped bit to complete the error correction process. One important thing is that these measurements ( $ZZ$ ) does not spoil the superposition of quantum states that we wish to preserve since they never give any information about  $a, b$  (amplitudes). The bit flip code is similar in spirit to the classical error correcting codes. One more advanced code is the ‘phase-flip’ code which has no classical counterpart and hence is more interesting as a quantum code. But note that there are ways to describe ‘phase-flip’ into ‘bit-flip’. We can always go from ‘quantum’ to ‘classical’ but not the other way round!

## 6.2 Phase-flip error

We now discuss another error that can occur. Instead of the operator  $X$  which was applied to the bit flip case, consider now the action of  $Z$  operator. Then the state  $a|0\rangle + b|1\rangle$  is transformed to

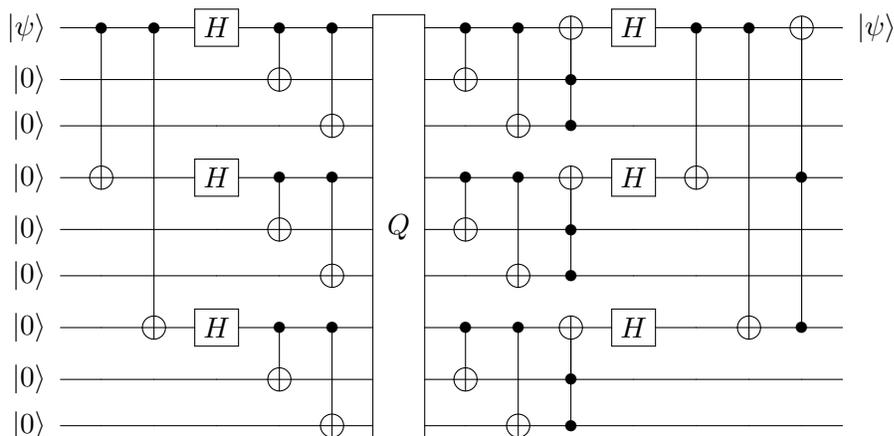
$a|0\rangle - b|1\rangle$ . Now suppose we rather work in the basis  $|\pm\rangle$ , then  $Z|+\rangle \rightarrow |-\rangle$ . Hence, phase flip is like bit flip in a different basis! This presents itself that all the basis parts of the error-correction i.e., encoding, error-detection, and recovery can be applied as bit flip code. The encoding circuit for bit flip will then be:



**Figure 12.** Circuit to encode the phase flip code.

### 6.3 Shor's nine qubit code

Though it is nice to correct the bit and phase flip separately, one desires a code to correct any arbitrary error because of these two using a single code. The ability to correct arbitrary errors on a *single* qubit can be achieved by Shor's nine-qubit code and is shown in Fig. 13. The basic step



**Figure 13.** The circuit to implement Shor's code where  $Q$  is a channel which can corrupt single qubit.

of Shor's construction is to note that:

$$|0\rangle = |0\rangle_L = |+++ \rangle, \quad (6.3)$$

$$|1\rangle = |1\rangle_L = |-- - \rangle, \quad (6.4)$$

will control the phase errors, while,

$$|+\rangle = \frac{1}{\sqrt{2}} (|000\rangle + |111\rangle), \quad (6.5)$$

$$|-\rangle = \frac{1}{\sqrt{2}} (|000\rangle - |111\rangle), \quad (6.6)$$

will correct against the bit flip error. Combining these two, we can write:

$$|0\rangle \rightarrow \frac{1}{2\sqrt{2}} \left( |000\rangle + |111\rangle \right) \otimes \left( |000\rangle + |111\rangle \right) \otimes \left( |000\rangle + |111\rangle \right). \quad (6.7)$$

Suppose the error is modelled by a unitary transformation  $U$  such that:

$$U = c_0\mathbb{1} + c_1X + c_2Y + c_3Z \quad (6.8)$$

Now, if  $U$  is equal to  $\mathbb{1}$ , then no error has occurred. If  $U = X$ , then a bit flip occurs, if  $U = Z$  then a phase flip occurs. Lastly, if  $U = iY$  then both a bit flip error and a sign flip error occur.

► **QUESTION 19:** Show that if the probability of single qubit being affected (either phase or bit flip or mixture of the two) is  $p$ , then the probability of having more than one qubit having error in Shor's nine qubit code is  $36p^2$ .

The quantum error-correcting conditions are a set of equations which have to be satisfied by a quantum error-correcting code to protect against a particular type of noise. We will do a short example which will clarify this for the reader. Suppose we are asked whether the three qubit phase flip code  $|0_L\rangle = |+++ \rangle$ ,  $|1_L\rangle = |-- - \rangle$ . satisfied the conditions for the set of error operators  $\{I, Z_1, Z_2, Z_3\}$ . The stabilizer set of Shor's code is<sup>29</sup>:

$$\{Z_1Z_2, Z_2Z_3, Z_4Z_5, Z_5Z_6, Z_7Z_8, Z_8Z_9, X_1X_2X_3X_4X_5X_6, X_4X_5X_6X_7X_8X_9\} \quad (6.9)$$

The Shor's code provides the window to the more complicated quantum error-correcting (QEC) codes. However, this code is not the most optimal and Steane 7-qubit code [61] is better. This code has six stabilizer generators compared to eight in Shor's code. These are given by:

$$\{IIIXXXX, IXXIIXX, XIXIXIX, IIIZZZZ, IZZIIZZ, ZIZIZIZ\} \quad (6.10)$$

All these QEC codes are best explained by the idea of stabilizer codes. For example, one famous example is the toric code, which is a topological quantum error correcting code and is an example of a stabilizer code/circuit<sup>30</sup>. It is defined on a two-dimensional spin lattice and has  $\mathbb{Z}_2$  topological order. It reduces to a  $\mathbb{Z}_2$  lattice gauge theory well known for decades in an appropriate limit. In this code, the stabilizer operators are defined on the spins around each vertex and on the plaquette and are known as  $A$  and  $B$  operators defined as:  $A_v = \prod_{i \in v} \sigma_i^x$  and  $B_p = \prod_{i \in p} \sigma_i^z$  respectively. We often refer to a QEC code as  $[[n, k, 2t + 1]]$  code<sup>31</sup> and Shor's code is an example of  $[[9, 1, 3]]$  stabilizer code. Sometimes,  $2t + 1$  is also called 'distance', and code is referred to as  $[[n, k, d]]$  code. Soon after nine-qubit code, Steane found a seven-qubit code which can correct the same error as

<sup>29</sup>Note that  $Z_1Z_2 = ZZIIIIII$  or  $Z \otimes Z \otimes I_{128}$

<sup>30</sup>A stabilizer circuit is a quantum circuit in which every gate is selected from a stabilizer set i.e.,  $\{H, \text{CNOT}, P\}$  or a 1-qubit measurement gate

<sup>31</sup>The two braces denote that is is a quantum code unlike classical code such as Hamming code which uses single parentheses

Shor’s code. In fact, both these codes are ‘degenerate’ which loosely means that it can be further improved. One of the ways we can optimize the code is to introduce the quantum Hamming bound [62]. If a given code uses  $n$  qubits, corrects error on up to  $t$  qubits and encodes  $k$  qubits, then the following inequality holds:

$$\left( \sum_{j=0}^t 3^j \binom{n}{j} \right) 2^k \leq 2^n. \quad (6.11)$$

It is easy to check that with  $k = 1, t = 1$ , the lower bound is saturated with  $n = 5$  hence there exists a 5-qubit code which is  $[[5,1,3]]$  which is nondegenerate. In general, it is clear that the stabilizer circuits/codes are very important for performing quantum error-correction and play important role in fault tolerant circuits. The idea of stabilizer circuits is also important for a theorem known as ‘Gottesman-Knill’ (GK) theorem[63]. The theorem implies that any stabilizer circuits can be perfectly simulated in polynomial time on a (probabilistic) classical computer. Hence, they offer no quantum advantage by themselves. This theorem originated in Ref. [63] where author refers to private communication with Knill. This result does not explain the reason why quantum computing can provide exponential speedup over classical computing, but it does tell us that quantum algorithms which make use of entanglement built using Hadamard and CNOT gates do not offer any advantage. This result is contrary to natural expectation that no quantum gates can be simulated efficiently by classical computers, since  $n$ - qubit quantum circuit operates in a  $2^n$ -dimensional Hilbert space. But admittedly the situation is very different for non-Clifford gates which cannot be simulated efficiently. If it was shown someday that even non-Clifford gates can be simulated by classical computers, then it would imply further (and significant) lack of usefulness of quantum computers.

## SECTION 7

### Looking at NISQ and beyond

---

One of the major final goal of building the quantum computer, at least for physicists, is to solve the time-evolution of some complicated quantum mechanical system. However, it is clear that a major portion of theories of interest cannot be simulated on NISQ devices with just few hundreds of noisy qubits. However, there is still a lot to understand and this is active area of research. It is already evident that even for non-relativistic quantum systems the size of Hilbert space is very large and adding the relativistic behaviour when studying quantum field theories make it harder. The infinite-dimensional space makes no sense for digital computers. One has to understand the reduction of the degrees of freedom properly keeping the symmetries (such as ‘gauge symmetry’) intact. There have been various proposals such as D-theory/quantum link models [64], dual variable using character expansions with sum over irreducible representations of the gauge group [65] which are then truncated, quantum groups such as  $SU(2)_q$  approach, loop-string-hadron formalism [66], light front dynamics method, and truncating the continuous symmetry groups by discrete point symmetry such as in Ref. [67]. All these methods are based on finding smaller subspace of the

vector space to be represented by qubits. There is also a method based on continuous variables (CVs) which plays an important role in analog quantum computing and has several advantages over the discrete methods. These schemes have their merits and drawbacks and the effectiveness in obtaining the continuum limit are still debatable. Some qubit regularization schemes also exist for  $O(3)$  non-linear sigma model by exploiting equivalence to  $SU(2)$  and associated truncation over irreducible representations but it is by no means a general method and works only for this specific group. We do not know of any work which attempts  $O(N)$  models with  $N \geq 4$ . Even for scalar field theories such as  $\phi^4$  theory in 1+1-dimensions, the quantum simulation and extracting of real-time properties and correlators are still not fully understood. There are approaches based on field variables following JLP [68], harmonic oscillator basis [69], continuous variable approach [70], and single-particle basis [71] to name a few. Here, we will simply focus on some simple example where one desires time evolution using some Hamiltonian with circuit depths<sup>32</sup> that may be performed on the NISQ devices.

We can perceive any quantum circuit as a simulation of some Hamiltonian  $H$ . This follows from the standard observation that any (noiseless) quantum circuit can be built up of pieces of unitary transformations whose product is also unitary. Once we have a given unitary transformation, we can find  $H$  such that  $U = e^{-iHt}$  where  $t$  is some real parameter. The time evolution of any given state can be defined by this  $U$ . This same idea (which goes by the name ‘Hamiltonian (hermitian) simulation’) plays an important role in different problems such as in the HHL algorithm [72] which is a quantum algorithm to solve linear system of equations. For example, if we had a term like  $\sigma_z \otimes \sigma_z$  in the Hamiltonian, then we could express  $\exp(i\sigma_z \otimes \sigma_z t)$  as  $e^{i\sigma_z \otimes \sigma_z t} = \cos(t)\mathbb{1} + i \sin(t)\sigma_z \otimes \sigma_z$ <sup>33</sup> which can be written by some manipulation (show this) as  $\text{CNOT}(\mathbb{1} \otimes e^{i\sigma_z t})\text{CNOT}$  since  $\text{CNOT} = |0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes \sigma_x$ . The circuit to implement [73] this term is given below:



where the argument of  $R_z$  gate is  $-2t$ . Similarly, we can construct a circuit for the three qubit term,  $\sigma_z \otimes \sigma_z \otimes \sigma_z$  as:

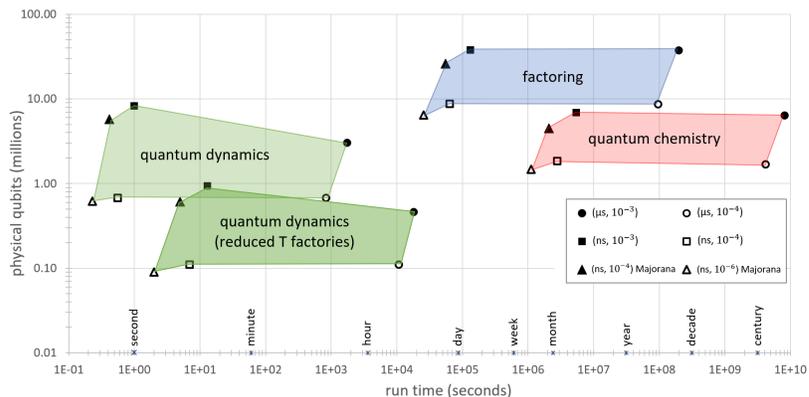


Sometimes, for brevity, we refer to the Pauli matrices just by  $X, Y, Z$  respectively. Now, suppose we have different Pauli matrices term in the Hamiltonian as:  $(X \otimes X + Y \otimes Y)$ . The corresponding circuit can be used to perform time evolution i.e., it is equivalent to  $\exp[i(\alpha/2)(X \otimes X + Y \otimes Y)]$

<sup>32</sup>The depth of a circuit is the longest path between the data input and the output, where each intermediate gate counts as one. The circuit depth can be easily found in QISKIT using `qc.depth()` where `qc` is the quantum circuit.

<sup>33</sup>Note that we have  $\exp(\pm i\theta U) = \cos(\theta)\mathbb{1} \pm i \sin(\theta)U$  if  $U^2 = \mathbb{1}$ .





**Figure 14.** Estimate of the number of qubits needed to perform different types of computations. The inset mentions the effect of gate operation time on the estimate. The figure is taken from Ref. [74].

SECTION 9

## Acknowledgements

The research was supported by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Co-design Center for Quantum Advantage under contract number DE-SC0012704. Research at Perimeter Institute was supported by the Government of Canada through the Department of Innovation, Science and Economic Development Canada and by the Province of Ontario through the Ministry of Research, Innovation and Science. All the quantum circuits in this article were made using `Qcircuit` introduced in Ref. [75].

SECTION A

## Some examples using Qiskit

As mentioned before, QISKIT (pronounced kiss-kit) is a backend simulator created by IBM where we can build up a code which can then be sent to actual quantum hardware or quantum processing units (QPU) in use by companies like IBM, Amazon, Honeywell etc. This is like how commercial airline pilots train in a simulator when a new Boeing/Airbus aircraft is about to be launched<sup>34</sup>. Once we have sufficient practice, we can pass the quantum program to the real machines, but until we really need them for some problem, the simulator is good enough to learn and get acquainted with the fundamental working of the computation. We also note that the calculation of computation time and associated charges for quantum computing on real devices is not as straightforward as classical computers. For example, we will see below that to get reasonable output from the

<sup>34</sup>In fact, one of the reasons for the failure of 737 Max aircraft resulting in tragic accidents around 2019 is that even though some new systems were introduced, the pilots were never ordered to take the new simulator training because Boeing cited similarity with the ones they had already taken. It was however not true, and the cause of accidents (MCAS system) was indeed the crucial difference.

simulator, we have to perform a certain step number of times, which is referred to as ‘shot’ i.e., a shot is a single execution of a quantum algorithm on a QPU. Depending on the quality of QPU, the per shot cost is determined. So, it might be that some quantum computing hardware company will charge 1 for 100 shots and some will charge the same for 1000 shots. The per-shot price is usually not affected by the number or type of gates used in a quantum circuit or the number of variables employed. The idea of shots is closely related to the errors in quantum computers, and it is to be expected that if there is a QPU with a well-defined error correcting procedure, then it will limit the number of shots needed considerably. Another part of the computation cost is based on number of ‘tasks’. A task is a sequence of successive shots using the same circuit design. QISKIT is made up of four elements, each of which has a specific functionality. These elements are Terra (Earth), Aer (Air), Ignis (Fire), and Aqua (Water). For example, the Aer provider contains a variety of high performance simulator backends for a variety of simulation methods. The available backends on the current system can be viewed using `Aer.backends()` in a Jupyter notebook.

This open-source software platform enables one to work with the quantum language Open Quantum Assembly Language (Open QASM), used in quantum computers in the IBM. We encourage the reader to read more about them if interested in Ref. [47]. Here, we will briefly summarize the different core modules of QISKIT for convenience. The Qiskit open-source development library is composed of four core modules:

- Qiskit Terra: This contains core elements to create quantum programs at the circuit/pulses level as well as to optimize these core elements by considering the particular physical quantum processor constraints.
- Qiskit Aer: This provides a C++ simulator framework and tools to develop noise models to conduct realistic simulations in the presence of noise and errors occurring during execution on real quantum devices.
- Qiskit Ignis: This provides a framework to understand the noise sources in the quantum circuits/devices as well as to develop different noise reduction procedures.
- Qiskit Aqua: This contains a library of cross-domain quantum algorithms suitable for application in the NISQ era.

The syntax of QISKIT consists of following high-level steps: 1. Building step in which we design a quantum circuit that represents the problem at hand, 2. The execution step in which we run experiments on different backends (including both actual quantum hardware and simulators), and the last is the analysis step where one summarizes the statistics and visualize the results. The SDK we will focus on in these notes deals with systems with finite local Hilbert space of  $d = 2$  (qubits) or maybe some generalization to qudits too ( $d > 2$ ) but they are often not the best way for bosonic states which have infinite sized vector space. For these models, it is often more practical to make use of what is called *qumodes* which are continuous-variable quantum modes. Very recently, an open-source software was introduced in Ref. [76] which extends QISKIT to admit bosonic degrees of freedom. This enables the simulation of hybrid qubit and bosonic systems

such as bosonic Hubbard models, Jaynes-Cummings etc. However, the support for parametrized circuits<sup>35</sup> is still in development phase. Another open-source project which is probably better suited for ground state energy computation of bosonic models is Xanadu’s **Strawberry Fields**, which is a full-stack Python library for designing, optimizing, and utilizing photonic quantum computers (analog computing). Using continuous variables (CV) instead of the more traditional qubits (digital) approach has both advantages and disadvantages. Some groups [55] have explored the use of QITE (quantum imaginary-time evolution) to study the ground state energy using CVs. This approach might also be useful in the future for other interesting models such as  $\phi^4$ -theory. Though the scope of this discussion lies beyond the scope of the current article, we will briefly mention some differences to qubit approach. The CV approach to quantum computing keeps the same computational power of the qubit model. This approach utilizes the infinite-dimensional bosonic mode called ‘qumodes’. While the qubits make use of discrete coefficients, the CV model makes use of bosonic harmonic oscillator, which is defined by the operators  $a$  and  $a^\dagger$  satisfying  $[a, a^\dagger] = 1$  and where  $|x\rangle$  are the eigenstates of  $x$  operator defined in terms of  $a, a^\dagger$ . In this approach, the information about input states are stored in terms of Gaussian states which are non-classical states with Gauss-Wigner quasi-probabilistic functions.

### A.1 Installing package and quantum hello world

In this section, we will provide several codes related to sections in the main text. The easiest way is to visit Google’s Colab<sup>36</sup> and open a new Jupyter notebook. We can install QISKIT by doing: `!pip install qiskit ipywidgets`. A simple check that everything is installed properly is to write this short piece of code and run it.

```

1  from qiskit import *
2  qc = QuantumCircuit(2); # Circuit of two qubits
3  # This is equivalent to QuantumCircuit(2,0) i.e., 2 qubits and no classical bits
4  qc.h(0); # Apply Hadamard gate to the first qubit
5  qc.draw() # Draw the circuit
6

```

We then extend this to write our first ‘quantum’ code given below:

```

1  # Welcome to "Hello World" for quantum computing
2  from qiskit import *
3  from qiskit import QuantumCircuit, assemble, Aer
4  from qiskit.visualization import plot_bloch_multivector, plot_histogram
5  sim = Aer.get_backend('aer_simulator')
6
7  def x_measurement(qc, qubit, cbit):

```

<sup>35</sup>Parameterized quantum circuits consist of quantum gates defined through tunable parameters. They are fundamental building blocks of majority of near-term quantum machine learning algorithms

<sup>36</sup><https://research.google.com/colaboratory>

```

8     qc.h(qubit) # Apply Hadamard to get superposition
9     qc.measure(qubit, cbit)
10    # Measure 'qubit' in the X-basis, and store the result in 'cbit
11    return qc
12
13    qc = QuantumCircuit(1,1); # Initialize one quantum and one classical bit.
14    x_measurement(qc, 0, 0)
15    qobj = assemble(qc) # Assemble circuit to run
16    counts = sim.run(qobj).result().get_counts() # Do the simulation, returning the state vector
17    print (counts) # Count the occurrences
18    plot_histogram(counts) # Display the output as a histogram
19

```

## A.2 Some basic operations using qubits and gates

Let us initialize two qubits in  $|0\rangle$  each. We can apply Hadamard gate to the first qubit and then feed them through CNOT gate. It is easy to check that it generates the output as:  $(1/\sqrt{2})(|00\rangle + |11\rangle)$ . We provide the code below to compute this out state and check that it agrees with the expectations. We leave this to the reader.

```

1  from qiskit import *
2  from qiskit.quantum_info import Statevector
3  from qiskit.visualization import plot_bloch_multivector, plot_histogram, array_to_latex
4  qc = QuantumCircuit(2);
5  qc.h(0);
6  qc.draw()
7  q2 = qc.cx(0,1);
8  bell = Statevector.from_instruction(qc)
9  display(array_to_latex(bell, prefix="\text{Statevector} = "))
10

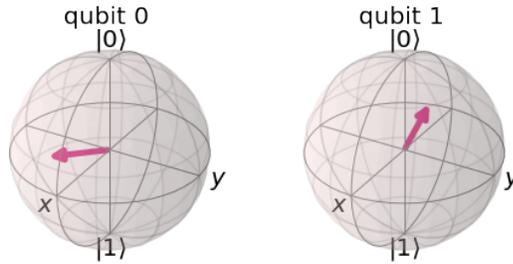
```

It is easy to generate  $d = 2^n$ -dimensional state vector using a short program and then view it on the Bloch sphere. Suppose we take  $n = 2$  (two qubits) and generate a random state  $|\psi\rangle$ . It can be done as given below. Running this different times will move the arrow (on the Bloch sphere) corresponding to different random state. The reader is encouraged to check that they are appropriately normalized, i.e.,  $\langle\psi|\psi\rangle = 1$ . The output on the Bloch sphere for a random state looks like that given in Fig. 15. Note that orthogonal states (such as  $|0\rangle$  and  $|1\rangle$ ) are antipodal on the Bloch sphere.

```

1  from qiskit.quantum_info import random_statevector, Statevector
2  from qiskit.visualization import plot_bloch_multivector
3
4  rand_sv = random_statevector(4).data
5  print(rand_sv)
6  plot_bloch_multivector(rand_sv)
7

```



**Figure 15.** A random two-qubit state depicted on the sphere.

Now, since we have a bipartite random state, we can compute EOF as discussed in Sec. 3.3.2. For example, if we compute this for Bell state mentioned above, it will be 1 (maximally entangled). We give the code below:

```

1 from qiskit import *
2 from qiskit.quantum_info import *
3
4 rand_sv = random_statevector(4).data
5 print("Entanglement of Formation = ", entanglement_of_formation(rand_sv))
6

```

► **QUESTION 21:** Set up a quantum circuit using QISKIT to create the state given by Eq. 3.3 and show that it is not entangled.

Let us now show the program to implement CSWAP gate as discussed in the text using QISKIT. Note that if we take  $q_0 = |+\rangle$ ,  $q_1 = |0\rangle$ , and  $q_2 = |1\rangle$ , then the action of CSWAP looks like:

$$|1\rangle \otimes |0\rangle \otimes (1/\sqrt{2})(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}(|100\rangle + |101\rangle) \xrightarrow{\text{CSWAP}} \frac{1}{\sqrt{2}}(|100\rangle + |011\rangle) \quad (\text{A.1})$$

```

1 from qiskit import *
2 from qiskit.quantum_info import *
3 from qiskit.visualization import *
4
5 qc = QuantumCircuit(3)
6 qc.h(0);
7 qc.x(2);
8 st = Statevector.from_instruction(qc)
9 display(array_to_latex(st, prefix="\text{In state} = "))
10 qc.cswap(0,1,2)
11 st = Statevector.from_instruction(qc)

```

```

12 display(array_to_latex(st, prefix="\\text{ Out state} = "))
13 qc.draw()
14

```

In the program above, we have used `qc.cswap(0,1,2)` to implement CSWAP. Suppose we do not have this three qubit gate and only have access to CNOT gate. We encourage the reader to write the equivalent program. We can also entangle  $n$  qubits (using GHZ state construction) as follows:

```

1  from qiskit import *
2  from qiskit.visualization import *
3  from qiskit.quantum_info import *
4
5  # We will now construct GHZ state
6  qn = 5 # five qubit GHZ
7
8  circ = QuantumCircuit(qn)
9  circ.h(0)
10 circ.cx(0, 1)
11 circ.cx(0, 2)
12 circ.cx(0, 3)
13 circ.cx(0, 4)
14 circ.draw()
15 ghz = Statevector.from_instruction(circ)
16 display(array_to_latex(ghz, prefix="\\text{Statevector} = "))
17

```

Suppose we prepare some initial state and someone applies a quantum gate in our absence, is there a way in QISKIT to see what this transformation was? The way to do it, this is (apart from usual stuff):

```

1  job = execute(qc, Aer.get_backend('unitary_simulator'))
2  result = job.result()
3  print(result.get_unitary(qc, decimals=3))
4

```

We now turn to the computation of fidelity of two states which was discussed in the main text. Consider two-states given by:

$$|\psi\rangle = \frac{1}{2} \left( |001\rangle + |101\rangle - |011\rangle - |111\rangle \right),$$

and

$$\frac{1}{\sqrt{2}} \left( |000\rangle + |111\rangle \right).$$

Compute the fidelity between them using QISKIT.

```

1  from qiskit import *
2  from qiskit.quantum_info import *
3  from qiskit.visualization import *
4  from numpy import *
5
6  qc1 = QuantumCircuit(3)
7  qc1.h(0);
8  qc1.h(1);
9  qc1.x(2);
10 qc2 = QuantumCircuit(3)
11 qc2.h(0)
12 qc2.cx(0,1);
13 qc2.cx(0,2);
14
15 backend = Aer.get_backend('statevector_simulator')
16 sv1 = execute(qc1, backend).result().get_statevector(qc1)
17 sv2 = execute(qc2, backend).result().get_statevector(qc2)
18 print(state_fidelity(sv1, sv2))
19

```

### A.3 Quantum full-adder

Similar to the classical logic gates where one can implement full-adder, we can do similar operation using quantum gates. The code is given below.

```

1  # Quantum full-adder program
2
3  from qiskit import *
4  from qiskit.visualization import *
5
6  qc = QuantumCircuit(4,2)
7
8  # Preparing inputs
9  # We do: A and B to a Bell state and CarryIn to a superposition state
10 qc.h(0)
11 qc.cx(0,1)
12 # To Bell State: Hadamard followed by CNOT
13 qc.h(2) # CarryIn to superposition by Hadamard
14
15 # Adder starts
16 qc.ccx(0,1,3) # Toffoli gate [three qubit]
17 qc.cx(0,1)
18 qc.ccx(1,2,3)
19 qc.cx(1,2)
20 qc.cx(0,1)
21
22 qc.measure(2,0) # Sum

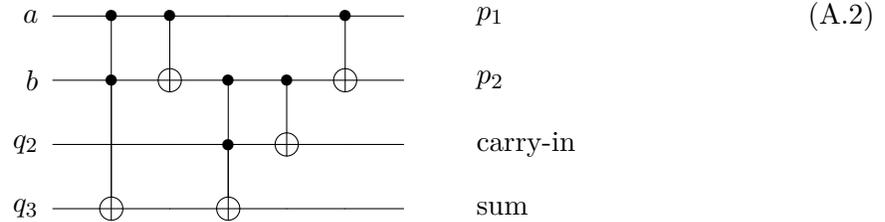
```

```

23 qc.measure(3,1) # Carry-out
24 counts = execute(qc, Aer.get_backend('qasm_simulator'), shots=1024).result().get_counts()
25 print(counts)
26 plot_histogram(counts)

```

The circuit for quantum full-adder is given by (we have left out the part where qubits are initialized, see code for details):



#### A.4 Grover's algorithm with general diffuser routine

We give the QISKIT code for Grover's algorithm for the interested reader. We implement two cases of oracle for finding the target state  $|11\rangle$  and  $|110\rangle$  as example. The diffuser part (implementing  $D$ ) is general for any number of qubits.

```

1  from qiskit import *
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  n = 3
6
7  def diffuser(nqubits):
8
9      # D = (H)^n . (X)^n . (MCZ) . (X)^n . (H)^n
10     qc = QuantumCircuit(nqubits)
11
12     # Apply transformation |s> -> |0 ... 0> (H-gates)
13     for qubit in range(nqubits):
14         qc.h(qubit)
15
16     # Apply transformation |0 ... 0> -> |1 ... 1> (X-gates)
17     for qubit in range(nqubits):
18         qc.x(qubit)
19
20     # *****
21     # Do multi-controlled-Z gate
22     # by using Multi controlled Toffoli
23     # Recall that HXH = Z
24     qc.h(nqubits-1)
25     qc.mct(list(range(nqubits-1)), nqubits-1) # Multi-controlled
26     qc.h(nqubits-1)
27     # *****

```

```

28
29 # Now X to all
30 # Apply transformation |1 ... 1> -> |0 ... 0>
31 for qubit in range(nqubits):
32     qc.x(qubit)
33
34 # Apply transformation |0 ... 0> -> |s>
35 for qubit in range(nqubits):
36     qc.h(qubit)
37
38 # We will return the diffuser as a gate
39 U_s = qc.to_gate()
40 U_s.name = "Diffuser (D)"
41 return U_s
42
43
44 oracle = QuantumCircuit(n, name = 'Oracle (0)')
45
46 if n == 2:
47     oracle.cz(0,1)
48 if n == 3:
49     oracle.cz(1, 2)
50
51 oracle.to_gate()
52 # Oracle is CZ gate. This will change based on target state.
53 # We provide two example |11> and |110>
54
55 backend = Aer.get_backend('statevector_simulator')
56 gro_circ = QuantumCircuit(n,n)
57 gro_circ.h(list(range(n)))
58 gro_circ.append(oracle,list(range(n)))
59
60 gro_circ.append(diffuser(n),list(range(n)))
61
62 backend = Aer.get_backend('qasm_simulator')
63 gro_circ.measure([0,1],[0,1])
64 job = execute(gro_circ,backend, shots=1)
65 result = job.result()
66 result.get_counts()

```

## A.5 Implementing QFT

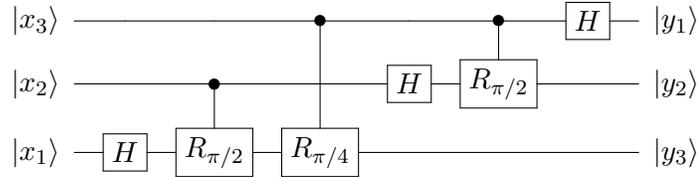
We now discuss the implementation of quantum Fourier transform in QISKIT. This makes use of two quantum gates, which we have already seen a lot of. The first is the Hadamard gate, the other is the controlled phase gate  $R$ . The matrix representation of these gates are given below:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad (\text{A.3})$$

while the rotation matrices are interchangeably defined as both:

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & \exp\left(\frac{2\pi i}{2^k}\right) \end{bmatrix}, \quad R_\theta = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix} \quad (\text{A.4})$$

The simplified QFT circuit for three qubits is given below, while the program in QISKIT below is for general  $n$ -qubit case.



The code is given below:

```

1  # QuantumFourier program
2
3  from qiskit import *
4  from numpy import *
5  from qiskit.quantum_info import *
6  from qiskit.visualization import *
7
8
9  def qft_rotations(circuit, n):
10
11     if n == 0:
12         return circuit
13
14     n -= 1
15     circuit.h(n)
16
17     for qubit in range(n):
18         circuit.cp(pi/2**(n-qubit), qubit, n)
19
20     qft_rotations(circuit, n)
21
22
23  def swap_registers(circuit, n):
24
25     for qubit in range(n//2):
26         circuit.swap(qubit, n-qubit-1)
27     return circuit
28
29
30  def qft(circuit, n):
31
32     qft_rotations(circuit, n) # Goes in a circle until all done
33     swap_registers(circuit, n)

```

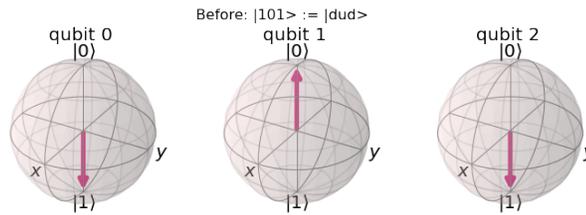


Figure 16. The initial state  $|101\rangle$  taken as example in the code in Section XXX

```

34     return circuit
35
36
37 nbits = 3 # number of qubits
38 N = 2**nbits
39 # So for this case it will be QFT_8
40
41 qc = QuantumCircuit(nbits)
42
43 # If done nothing, encodes 000 [i.e., all qubits set to 0 i.e., |000>]
44 # We will try encoding 5 i.e., |101>
45 qc.x(0)
46 qc.x(2)
47
48 sim = Aer.get_backend("aer_simulator")
49 qc_init = qc.copy()
50 qc_init.save_statevector()
51 statevector = sim.run(qc_init).result().get_statevector()
52 plot_bloch_multivector(statevector, title= "Before: |101> := |dud> ")
53
54 # Apply QFT
55
56 qft(qc,nbits)
57 #qc.draw()
58
59 psi_AB = Statevector.from_instruction(qc)
60 plot_bloch_multivector(psi_AB, title= "After: QFT_8|101>")

```

## A.6 Bit-flip code, Phase kickback

```

1 # A simple example of how the bit flip of q0 can be corrected
2 # This version can be modified to do bit flip with probability 'p' < 1
3
4 from qiskit import *
5 from qiskit import Aer
6
7 backend = Aer.get_backend('qasm_simulator')

```

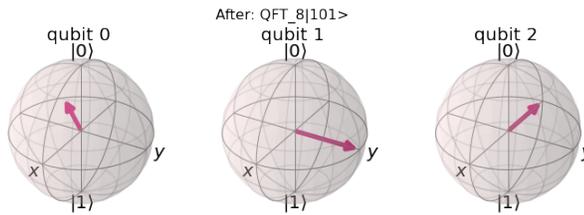


Figure 17. The state after the transformation.

```

8  circuit = QuantumCircuit(3,1)
9
10 # Encode
11 circuit.cx(0,1)
12 circuit.cx(0,2)
13
14 # Noise
15 circuit.x(0) # Impose a bit flip error by Pauli X
16
17 # Decode
18 circuit.cx(0,1)
19 circuit.cx(0,2)
20 circuit.ccx(2,1,0)
21 circuit.measure(0,0)
22
23 job = execute(circuit, backend, shots=1024)
24 counts = job.result().get_counts()
25 print(counts)

```

Now we do phase kickback to reproduce the state mentioned in (4.2) with  $\phi = \pi/4$ .

```

1  # Phase kickback
2
3  from qiskit import *
4  from numpy import *
5  from qiskit.quantum_info import Statevector
6  from qiskit.visualization import plot_bloch_multivector, plot_histogram, array_to_latex
7
8  def phase_kickback():
9      qreg = QuantumRegister(2, 'q')
10     qc1 = QuantumCircuit(qreg)
11     qc1.x(qreg[1])
12     qc1.h(qreg[0])
13     qc1.cp(pi/4, 0, 1) # Controlled T
14     return qc1
15
16 qc = phase_kickback()

```

```

17 qc.draw()
18 co = Statevector.from_instruction(qc)
19 plot_bloch_multivector(co)
20 # We can see the qubit 0 has been rotated by pi/4 around
21 # the Z-axis of the Bloch sphere as expected.

```

## A.7 VQE solution to anharmonic quantum oscillator for three qubits

In this section, we provide code which accompanies Sec. 5.1. We first decompose the Hamiltonian in terms of Paulis operators and then use it as input to VQE solver by defining some appropriate ansatz.

```

1  #!pip install qiskit ipywidgets
2  import numpy as np
3  import math
4  import time
5  import warnings
6  import itertools
7  from qiskit import Aer
8  from qiskit.algorithms import VQE
9  from qiskit.opflow import MatrixOp
10 from qiskit.opflow import X, Y, Z, I
11 from qiskit.algorithms.optimizers import SLSQP
12 from qiskit.algorithms.optimizers import COBYLA
13 from qiskit.circuit.library import EfficientSU2
14 from qiskit.utils import algorithm_globals, QuantumInstance
15 from qiskit.visualization.array import array_to_latex
16
17 def decompose(H):
18
19     def dagger(a):
20         return np.transpose(a).conj()
21
22     def HS(M1, M2):
23         # Hilbert-Schmidt prod. of two matrices
24         return np.trace(dagger(M1) @ M2)
25
26     def c2s(c):
27         if c == 0.0:
28             return "0"
29         if c.imag == 0:
30             return "%g" % c.real
31         elif c.real == 0:
32             return "%gj" % c.imag
33         else:
34             return "%g+%gj" % (c.real, c.imag)
35
36     sx = np.array([[0, 1], [1, 0]], dtype=np.complex128)

```

```

37 sy = np.array([[0, -1j],[1j, 0]], dtype=np.complex128)
38 sz = np.array([[1, 0], [0, -1]], dtype=np.complex128)
39 id = np.array([[1, 0], [0, 1]], dtype=np.complex128)
40 S = [id, sx, sy, sz]
41 labels = ['I', "X", "Y", "Z"]
42 final = ""
43
44 if nbits == 2:
45     for i, j in itertools.product(range(4),range(4)):
46         a = (1/2**nbits) * HS(np.kron(S[i], S[j]), H)
47         # Formula: (1/2^2) (Si otimes Sj) dot H
48         if a != 0.0:
49             term = c2s(a, '*', labels[i], '^', labels[j])
50             t2 = str(term).replace(', ', ' ')
51             t2 = str(t2).replace("'", "")
52
53             if final == "":
54                 final = "".join((final, t2))
55             else:
56                 final = "+".join((final, t2))
57
58 if nbits == 3:
59     for i, j, k in itertools.product(range(4),range(4),range(4)):
60         a = (1/2**nbits) * HS(np.kron(np.kron(S[i], S[j]),S[k]), H)
61
62         if a != 0.0:
63             term = c2s(a, '*', labels[i], '^', labels[j], '^', labels[k])
64             t2 = str(term).replace(', ', ' ')
65             t2 = str(t2).replace("'", "")
66
67             if final == "":
68                 final = "".join((final, t2))
69             else:
70                 final = "+".join((final, t2))
71
72     return final
73
74
75 # Alternative way
76
77 def numberToBase(n, b, n_qubits):
78     if n == 0:
79         return np.zeros(n_qubits,dtype=int)
80     digits = np.zeros(n_qubits,dtype=int)
81     counter = 0
82     while n:
83         digits[counter] = int(n % b)
84         n //= b

```

```

85     counter += 1
86     return digits[::-1]
87
88 def decomposePauli(H):
89
90     sx = np.array([[0, 1], [1, 0]], dtype=np.complex128)
91     sy = np.array([[0, -1j], [1j, 0]], dtype=np.complex128)
92     sz = np.array([[1, 0], [0, -1]], dtype=np.complex128)
93     id = np.array([[1, 0], [0, 1]], dtype=np.complex128)
94     S = [id, sx, sy, sz]
95     dim_matrix=np.shape(H)[0]
96     n_qubits=int(np.log2(dim_matrix))
97     if(dim_matrix != 2**n_qubits):
98         raise NameError("Matrix is not power of 2!")
99     hilbertspace = 2**n_qubits
100    n_paulis = 4**n_qubits
101    pauli_list = np.zeros([n_paulis,n_qubits],dtype=int)
102    for k in range(n_paulis):
103        pauli_list[k,:] = numberToBase(k,4,n_qubits)
104    weights = np.zeros(n_paulis,dtype=np.complex128)
105    for k in range(n_paulis):
106        pauli=S[pauli_list[k][0]]
107
108        for n in range(1,n_qubits):
109            pauli = np.kron(pauli,S[pauli_list[k][n]])
110            weights[k] = 1/hilbertspace* np.dot(pauli,H).trace()
111
112    nnz = [i for i, e in enumerate(weights) if e != 0]
113    for i in nnz:
114        print (weights[i],pauli_list[i])
115
116
117 # Yet another way
118 def decompose_ham_to_pauli(H):
119
120     n = int(np.log2(len(H)))
121     N = 2 ** n
122     # Basic checks!
123
124     if H.shape != (N, N):
125         raise ValueError("The Hamiltonian should have shape (2**n, 2**n), for any qubit
126         number n>=1")
127
128     if not np.allclose(H, H.conj().T):
129         raise ValueError("The Hamiltonian is not Hermitian")
130
131     sI = np.eye(2, 2, dtype=complex)
132     sX = np.array([[0, 1], [1, 0]], dtype=complex)

```

```

132 sZ = np.array([[1, 0], [0,-1]], dtype=complex)
133 sY = complex(0,-1)*np.matmul(sZ,sX)
134 paulis = [sI, sX, sY, sZ]
135 paulis_label = ['I', 'X', 'Y', 'Z']
136 obs = []
137 coeffs = []
138 matrix = []
139
140 for term in itertools.product(paulis, repeat=n):
141     matrices = [pauli for pauli in term]
142     coeff = np.trace(ft.reduce(np.kron, matrices) @ H) / N
143     coeff = np.real_if_close(coeff).item()
144
145     if not np.allclose(coeff, 0):
146         coeffs.append(coeff)
147         obs.append(''.join([paulis_label[[i for i, x in enumerate(paulis)
148             if np.all(x == t)][0]]+str(idx) for idx, t in enumerate(reversed(term))]))
149         matrix.append(ft.reduce(np.kron, matrices))
150
151     return obs, coeffs , matrix
152
153
154 g = 0.02 # Coupling used to create ham_H0.txt as input to this code
155 H = np.loadtxt("ham_H0.txt", dtype='f', delimiter='\t')
156 nbits = math.log2(np.shape(H)[0])
157
158 # Two options until nbits=3.
159
160 qubitOp = MatrixOp(primitive=H)
161 Hps = decompose(H)
162 print (Hps)
163
164 Hps = (4 * I ^ I ^ I)+(-0.152955 * I ^ I ^ X)+(-0.5 * I ^ I ^ Z)+(-0.12289 * I ^ X ^
    X)+(-0.0629948 * I ^ Y ^ Y)+(-1 * I ^ Z ^ I)+(0.0237627 * I ^ Z ^ X)+(-0.0280252 * X ^ I
    ^ X)+(-0.0561195 * X ^ X ^ X)+(0.0287333 * X ^ Y ^ Y)+(0.0107047 * X ^ Z ^ X)+(-0.0280252
    * Y ^ I ^ Y)+(-0.0287333 * Y ^ X ^ Y)+(-0.0561195 * Y ^ Y ^ X)+(0.0107047 * Y ^ Z ^
    Y)+(-2 * Z ^ I ^ I)+(0.0872346 * Z ^ I ^ X)+(0.0842295 * Z ^ X ^ X)+(0.041655 * Z ^ Y ^
    Y)+(0.0207442 * Z ^ Z ^ X)
165 # Nice formatting using LaTeX
166 # H_matrix = H.to_matrix()
167 # array_to_latex(H_matrix)
168
169 start_time = time.time()
170 var_form = EfficientSU2(qubitOp.num_qubits, su2_gates=['ry'], entanglement="full", reps=1)
171
172 rngseed = 5
173 warnings.filterwarnings("ignore")
174 backend = Aer.get_backend("statevector_simulator", max_parallel_threads=1,

```

```

    max_parallel_experiments=0)
175 q_instance = QuantumInstance(backend, seed_transpiler=rngseed, seed_simulator=rngseed)
176 #optimizer = SLSQP(maxiter=600)
177 optimizer = COBYLA(maxiter=600)
178 # COBYLA works little better!
179
180 # Run the VQE
181 vqe = VQE(ansatz=var_form,optimizer=optimizer,quantum_instance=q_instance)
182 ret = vqe.compute_minimum_eigenvalue(Hps)
183 vqe_result = np.real(ret.eigenvalue)
184 print("VQE Result:", vqe_result)
185 exact = 0.5 - (11/8)*g*g - (465/32)*g*g*g*g
186 print ("Exact result for cubic oscillator upto 0(g^4) is", exact)
187 print ("Error is", np.round(abs((exact-vqe_result)/exact)*100,10), "percent")
188 end_time = time.time()
189 runtime = end_time-start_time
190 print('Program runtime: ',runtime, "seconds")
191
192 # WCUBIC[n_, g_] := (n + 0.5) - ( 45 g^2)/12 (n^2 + n + 11/30);

```

It returns an output given below:

VQE Result: 0.4994476532684504

Exact result for cubic oscillator upto 0(g^4) is 0.499447675

Error is 4.3511e-06 percent

Program runtime: 2.118819236755371 seconds

## A.8 Phase estimation

We provide in this subsection the code to estimate the phase as discussed in the main text.

```

1  from qiskit import *
2  from qiskit.quantum_info import *
3  from qiskit.visualization import *
4  from numpy import *
5  import sys
6
7  qubits=4
8  cbits=3
9  theta= 2*pi*(1/4) # Actually \theta is without 2*pi but abuse notation!
10 qc = QuantumCircuit(qubits, cbits)
11
12 def qft_dagger(qc, n):
13
14     # Swap is crucial. Remove this to see change in result
15     for qubit in range(n//2):
16         qc.swap(qubit, n-qubit-1)
17         for j in range(n):

```

```

18         for m in range(j):
19             qc.cp(-pi/float(2**(j-m)), m, j)
20             qc.h(j)
21
22     for qubit in range(3):
23         qc.h(qubit)
24
25     qc.x(qbits-1) # Initialize |psi> to |1>
26
27     sheets = 1
28     for qubit_first_reg in range(qbits-1):
29         for i in range(sheets):
30             qc.cp(theta, qubit_first_reg, qbits-1); # Target is last qubit (i.e. first qubit of
31                 second register)
32             sheets *= 2
33
34     # Apply inverse QFT
35     qft_dagger(qc, qbits-1)
36     # Measure
37     qc.barrier()
38     for n in range(3):
39         qc.measure(n,n)
40
41     # Get results and plot
42
43     backend = Aer.get_backend('aer_simulator')
44     job = execute(qc,backend,shots=2048)
45     result = job.result()
46     count = result.get_counts()
47     plot_histogram(count)

```

## A.9 Sample Mathematica code for to compute vN entropy

We need to install the appropriate package (as discussed in Sec. 3.3.2) and then execute the commands below.

```

1  ent = Sqrt[1/2]*(KetV[0,2] \[CircleTimes] KetV[0,2]+ KetV[1,2] \[CircleTimes] KetV[1,2])
2  rho12 = DM[ent];
3  rhonew = {{5/12, 1/6, 1/6}, {1/6, 1/6, 1/6}, {1/6, 1/6, 5/12}};
4  vNEntropy[rhonew, {1}, {3}]*Log[2] // N
5  rho1 = PickRandomRho[4];
6  rho2 = PickRandomRho[4];
7  Chop[RelEnt[rho1, rho2]]
8  rstate = PickRandomRho[4]
9  vNEntropy[rstate, {1}, {4}] * Log[2]
10

```

## References

- [1] R. Landauer, “Information is physical,” *Physics Today* **44** no. 5, (May, 1991) 23–29. <https://doi.org/10.1063%2F1.881299>.
- [2] R. Landauer, “The physical nature of information,” *Physics Letters A* **217** no. 4-5, (Jul, 1996) 188–193. <https://doi.org/10.1016%2F0375-9601%2896%2900453-7>.
- [3] J. Preskill, “Quantum information and physics: some future directions,” *J. Mod. Opt.* **47** (2000) 127–137, [arXiv:quant-ph/9904022](https://arxiv.org/abs/quant-ph/9904022).
- [4] R. Landauer, “Irreversibility and heat generation in the computing process,” *IBM Journal of Research and Development* **5** no. 3, (Jul, 1961) 183–191. <https://doi.org/10.1147%2Frd.53.0183>.
- [5] C. H. Bennett, “The thermodynamics of computation—a review,” *International Journal of Theoretical Physics* **21** no. 12, (Dec, 1982) 905–940. <https://doi.org/10.1007%2Fbf02084158>.
- [6] P. Benioff, “The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines,” *Journal of Statistical Physics* **22** no. 5, (May, 1980) 563–591. <https://doi.org/10.1007%2Fbf01011339>.
- [7] P. Benioff, “Quantum mechanical hamiltonian models of turing machines,” *Journal of Statistical Physics* **29** no. 3, (Nov., 1982) 515–546.
- [8] C. H. Bennett, “Logical reversibility of computation,” *IBM Journal of Research and Development* **17** no. 6, (Nov, 1973) 525–532. <https://doi.org/10.1147%2Frd.176.0525>.
- [9] Y. Manin, “Computable and Uncomputable (in Russian),” 1980.
- [10] R. P. Feynman, “Simulating physics with computers,” *International Journal of Theoretical Physics* **21** no. 6, (1982) 467–488.
- [11] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th ed., 2011.
- [12] P. W. Shor, “Introduction to Quantum Algorithms,” *arXiv e-prints* (Apr., 2000) quant-ph/0005003, [arXiv:quant-ph/0005003](https://arxiv.org/abs/quant-ph/0005003) [quant-ph].
- [13] D. P. DiVincenzo, “The physical implementation of quantum computation,” *Fortschritte der Physik* **48** no. 9-11, (Sep, 2000) 771–783. <https://doi.org/10.1002%2F1521-3978%28200009%2948%3A9%2F11%3C771%3A%3Aaid-prop771%3E3.0.co%3B2-e>.
- [14] S. P. Jordan, H. Krovi, K. S. M. Lee, and J. Preskill, “BQP-completeness of Scattering in Scalar Quantum Field Theory,” *Quantum* **2** (2018) 44, [arXiv:1703.00454](https://arxiv.org/abs/1703.00454) [quant-ph].
- [15] S. Aaronson and L. Chen, “Complexity-theoretic foundations of quantum supremacy experiments,” 2016. <https://arxiv.org/abs/1612.05903>.
- [16] <https://ai.googleblog.com/2019/10/quantum-supremacy-using-programmable.html>.
- [17] P. A. M. Dirac, “A new notation for quantum mechanics,” *Mathematical Proceedings of the Cambridge Philosophical Society* **35** no. 3, (Jul, 1939) 416–418. <https://doi.org/10.1017%2Fs0305004100021162>.
- [18] R. P. Feynman, “Quantum mechanical computers,” *Foundations of Physics* **16** no. 6, (1986) 507–531.

- [19] W. Dür, G. Vidal, and J. I. Cirac, “Three qubits can be entangled in two inequivalent ways,” *Physical Review A* **62** no. 6, (Nov, 2000) . <https://doi.org/10.1103/PhysRevA.62.062314>.
- [20] P. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan, “A new universal and fault-tolerant quantum basis,” *Information Processing Letters* **75** no. 3, (Aug, 2000) 101–107. <https://doi.org/10.1016/S0020-0190%2800%2900084-3>.
- [21] A. Y. Kitaev, “Quantum computations: algorithms and error correction,” *Russian Mathematical Surveys* **52** no. 6, (Dec, 1997) 1191–1249. <https://doi.org/10.1070/Rm1997v052n06abeh002155>.
- [22] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, “Elementary gates for quantum computation,” *Phys. Rev. A* **52** (Nov, 1995) 3457–3467. <https://link.aps.org/doi/10.1103/PhysRevA.52.3457>.
- [23] C. M. Dawson and M. A. Nielsen, “The solovay-kitaev algorithm,” 2005. <https://arxiv.org/abs/quant-ph/0505030>.
- [24] D. P. DiVincenzo and J. Smolin, “Results on two-bit gate design for quantum computers,” *arXiv e-prints* (Sept., 1994) cond-mat/9409111, [arXiv:cond-mat/9409111](https://arxiv.org/abs/cond-mat/9409111) [cond-mat].
- [25] A. Barenco, “A Universal two bit gate for quantum computation,” *Proc. Roy. Soc. Lond. A* **449** (1995) 679, [arXiv:quant-ph/9505016](https://arxiv.org/abs/quant-ph/9505016).
- [26] Y. Shi, “Both Toffoli and Controlled-NOT need little help to do universal quantum computation,” *arXiv e-prints* (May, 2002) quant-ph/0205115, [arXiv:quant-ph/0205115](https://arxiv.org/abs/quant-ph/0205115) [quant-ph].
- [27] D. Aharonov, “A Simple Proof that Toffoli and Hadamard are Quantum Universal,” *arXiv e-prints* (Jan., 2003) quant-ph/0301040, [arXiv:quant-ph/0301040](https://arxiv.org/abs/quant-ph/0301040) [quant-ph].
- [28] S. Hill and W. K. Wootters, “Entanglement of a pair of quantum bits,” *Physical Review Letters* **78** no. 26, (Jun, 1997) 5022–5025. <https://doi.org/10.1103/PhysRevLett.78.5022>.
- [29] W. K. Wootters, “Entanglement of formation of an arbitrary state of two qubits,” *Physical Review Letters* **80** no. 10, (Mar, 1998) 2245–2248. <https://doi.org/10.1103/PhysRevLett.80.2245>.
- [30] W. K. Wootters, “Entanglement of formation and concurrence,” *Quantum Info. Comput.* **1** no. 1, (Jan, 2001) 27–44.
- [31] D. Coppersmith, “An approximate fourier transform useful in quantum factoring,” <https://arxiv.org/abs/quant-ph/0201067>.
- [32] W. K. Wootters and W. H. Zurek, “A single quantum cannot be cloned,” *Nature* **299** no. 5886, (Oct, 1982) 802–803. <https://doi.org/10.1038/2F299802a0>.
- [33] D. Dieks, “Communication by EPR devices,” *Physics Letters A* **92** no. 6, (Nov, 1982) 271–272.
- [34] A. K. Pati and S. L. Braunstein *Nature* **404** no. 6774, (Mar, 2000) 164–165. <https://doi.org/10.1038/2F35004532>.
- [35] D. Leach and A. Malvino, “Digital principles and applications,”.
- [36] D. Deutsch and R. Jozsa, “Rapid solution of problems by quantum computation,” *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* **439** no. 1907, (Dec, 1992) 553–558. <https://doi.org/10.1098/Rspa.1992.0167>.
- [37] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, “Quantum algorithms revisited,” *Proceedings of*

*the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* **454**  
no. 1969, (Jan, 1998) 339–354. <https://doi.org/10.1098%2Frspa.1998.0164>.

- [38] L. K. Grover, “A fast quantum mechanical algorithm for database search,” *arXiv e-prints* (May, 1996) quant-ph/9605043, [arXiv:quant-ph/9605043](https://arxiv.org/abs/quant-ph/9605043) [quant-ph].
- [39] L. K. Grover, “Quantum mechanics helps in searching for a needle in a haystack,” *Physical Review Letters* **79** no. 2, (Jul, 1997) 325–328. <https://doi.org/10.1103%2Fphysrevlett.79.325>.
- [40] P. W. Shor, “Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer,” *SIAM J. Sci. Statist. Comput.* **26** (1997) 1484, [arXiv:quant-ph/9508027](https://arxiv.org/abs/quant-ph/9508027).
- [41] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, “Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance,” *Nature* **414** no. 6866, (Dec, 2001) 883–887. <https://doi.org/10.1038%2F414883a>.
- [42] C. Gidney and M. Ekerå, “How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits,” *arXiv e-prints* (May, 2019) arXiv:1905.09749, [arXiv:1905.09749](https://arxiv.org/abs/1905.09749) [quant-ph].
- [43] M. Cerezo, A. Arrasmith, R. Babbush, S. Benjamin, S. Endo, K. Fujii, J. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles, “Variational quantum algorithms,” *nature reviews physics* (2020).
- [44] J. Tilly *et al.*, “The Variational Quantum Eigensolver: a review of methods and best practices,” [arXiv:2111.05176](https://arxiv.org/abs/2111.05176) [quant-ph].
- [45] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O’Brien, “A variational eigenvalue solver on a photonic quantum processor,” *Nature Communications* **5** no. 1, (Jul, 2014) . <https://doi.org/10.1038%2Fncoms5213>.
- [46] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, “Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets,” [arXiv:1704.05018](https://arxiv.org/abs/1704.05018) [quant-ph].
- [47] <https://qiskit.org/textbook/preface.html>.
- [48] B. F. Schiffer, J. Tura, and J. I. Cirac, “Adiabatic Spectroscopy and a Variational Quantum Adiabatic Algorithm,” *PRX Quantum* **3** no. 2, (2022) 020347, [arXiv:2103.01226](https://arxiv.org/abs/2103.01226) [quant-ph].
- [49] Raffaele Miceli and Michael McGuigan, “Quantum Computation and Visualization of Hamiltonians using Discrete Quantum Mechanics and IBM QISKIT,” [quant-ph/1812.01044](https://arxiv.org/abs/quant-ph/1812.01044).
- [50] O. Higgott, D. Wang, and S. Brierley, “Variational quantum computation of excited states,” *Quantum* **3** (Jul, 2019) 156. <https://doi.org/10.22331%2Fq-2019-07-01-156>.
- [51] H. Kleinert, “Path Integrals in Quantum Mechanics, Statistics, Polymer Physics, and Financial Markets,”.
- [52] E. Dumitrescu, A. McCaskey, G. Hagen, G. Jansen, T. Morris, T. Papenbrock, R. Pooser, D. Dean, and P. Lougovski, “Cloud quantum computing of an atomic nucleus,” *physical review letters* (2018) .
- [53] J. Y. Araz, S. Schenk, and M. Spannowsky, “Towards a Quantum Simulation of Nonlinear Sigma Models with a Topological Term,” [arXiv:2210.03679](https://arxiv.org/abs/2210.03679) [quant-ph].
- [54] M. Asaduzzaman, S. Catterall, G. C. Toga, Y. Meurice, and R. Sakai, “Quantum Simulation of the N flavor Gross-Neveu Model,” [arXiv:2208.05906](https://arxiv.org/abs/2208.05906) [hep-lat].

- [55] K. Yeter-Aydeniz, E. Moschandreou, and G. Siopsis, “Quantum imaginary-time evolution algorithm for quantum field theories with continuous variables,” *Phys. Rev. A* **105** no. 1, (2022) 012412, [arXiv:2107.00791 \[quant-ph\]](#).
- [56] S. McArdle, T. Jones, S. Endo, Y. Li, S. C. Benjamin, and X. Yuan, “Variational ansatz-based quantum simulation of imaginary time evolution,” *npj Quantum Information* **5** (Sept., 2019) 75, [arXiv:1804.03023 \[quant-ph\]](#).
- [57] M. Motta, C. Sun, A. T. K. Tan, M. J. O’Rourke, E. Ye, A. J. Minnich, F. G. S. L. Brandão, and G. K.-L. Chan, “Determining eigenstates and thermal states on a quantum computer using quantum imaginary time evolution,” *Nature Physics* **16** no. 2, (Jan., 2020) 205–210, [arXiv:1901.07653 \[quant-ph\]](#).
- [58] E. Knill and R. Laflamme, “A Theory of quantum error correcting codes,” *Phys. Rev. Lett.* **84** (2000) 2525–2528, [arXiv:quant-ph/9604034](#).
- [59] P. W. Shor, “Scheme for reducing decoherence in quantum computer memory,” *Phys. Rev. A* **52** (Oct, 1995) R2493–R2496. <https://link.aps.org/doi/10.1103/PhysRevA.52.R2493>.
- [60] R. Laflamme, C. Miquel, J. P. Paz, and W. H. Zurek, “Perfect quantum error correction code,” [arXiv:quant-ph/9602019](#).
- [61] A. Steane, “Multiple particle interference and quantum error correction,” *Proc. Roy. Soc. Lond. A* **452** (1996) 2551, [arXiv:quant-ph/9601029](#).
- [62] D. Gottesman, “An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation,” *arXiv e-prints* (Apr., 2009) arXiv:0904.2557, [arXiv:0904.2557 \[quant-ph\]](#).
- [63] D. Gottesman, “The Heisenberg representation of quantum computers,” in *22nd International Colloquium on Group Theoretical Methods in Physics*, pp. 32–43. 7, 1998. [arXiv:quant-ph/9807006](#).
- [64] S. Chandrasekharan and U. J. Wiese, “Quantum link models: A Discrete approach to gauge theories,” *Nucl. Phys. B* **492** (1997) 455–474, [arXiv:hep-lat/9609042](#).
- [65] A. Bazavov, S. Catterall, R. G. Jha, and J. Unmuth-Yockey, “Tensor renormalization group study of the non-Abelian Higgs model in two dimensions,” *Phys. Rev. D* **99** no. 11, (2019) 114507, [arXiv:1901.11443 \[hep-lat\]](#).
- [66] I. Raychowdhury and J. R. Stryker, “Loop, string, and hadron dynamics in SU(2) Hamiltonian lattice gauge theories,” *Phys. Rev. D* **101** no. 11, (2020) 114502, [arXiv:1912.06133 \[hep-lat\]](#).
- [67] A. Alexandru, P. F. Bedaque, R. Brett, and H. Lamm, “Spectrum of digitized QCD: Glueballs in a S(1080) gauge theory,” *Phys. Rev. D* **105** no. 11, (2022) 114508, [arXiv:2112.08482 \[hep-lat\]](#).
- [68] S. P. Jordan, K. S. M. Lee, and J. Preskill, “Quantum Computation of Scattering in Scalar Quantum Field Theories,” *Quant. Inf. Comput.* **14** (2014) 1014–1080, [arXiv:1112.4833 \[hep-th\]](#).
- [69] N. Klco and M. J. Savage, “Digitization of scalar fields for quantum computing,” *Phys. Rev. A* **99** no. 5, (2019) 052335, [arXiv:1808.10378 \[quant-ph\]](#).
- [70] K. Marshall, R. Pooser, G. Siopsis, and C. Weedbrook, “Quantum simulation of quantum field theory using continuous variables,” *Phys. Rev. A* **92** no. 6, (2015) 063825, [arXiv:1503.08121 \[quant-ph\]](#).
- [71] J. Barata, N. Mueller, A. Tarasov, and R. Venugopalan, “Single-particle digitization strategy for quantum computation of a  $\phi^4$  scalar field theory,” *Phys. Rev. A* **103** no. 4, (2021) 042410, [arXiv:2012.00020 \[hep-th\]](#).

- [72] A. W. Harrow, A. Hassidim, and S. Lloyd, “Quantum algorithm for linear systems of equations,” *Phys. Rev. Lett.* **103** (Oct, 2009) 150502.  
<https://link.aps.org/doi/10.1103/PhysRevLett.103.150502>.
- [73] J. D. Whitfield, J. Biamonte, and A. Aspuru-Guzik, “Simulation of electronic structure hamiltonians using quantum computers,” *Molecular Physics* **109** no. 5, (Mar, 2011) 735–750.  
<https://doi.org/10.1080%2F00268976.2011.552441>.
- [74] M. E. Beverland, P. Murali, M. Troyer, K. M. Svore, T. Hoeffler, V. Kliuchnikov, G. Hao Low, M. Soeken, A. Sundaram, and A. Vaschillo, “Assessing requirements to scale to practical quantum advantage,” *arXiv e-prints* (Nov., 2022) arXiv:2211.07629, [arXiv:2211.07629](https://arxiv.org/abs/2211.07629) [quant-ph].
- [75] B. Eastin and S. T. Flammia, “Q-circuit tutorial,” [arXiv:quant-ph/0406003](https://arxiv.org/abs/quant-ph/0406003) [quant-ph].
- [76] T. J. Stavenger, E. Crane, K. Smith, C. T. Kang, S. M. Girvin, and N. Wiebe, “Bosonic Qiskit,” [arXiv:2209.11153](https://arxiv.org/abs/2209.11153) [quant-ph].