Quantum computation with applications in seismic problems

Shahpoor Moradi and Daniel Trad

ABSTRACT

In this report, we motivate the need for quantum computation in seismic problems. We also developed a MATLAB package that simulates the well-known quantum algorithms for general n-qubit states. There are several software packages available to simulate the quantum algorithms on classical computers. These packages are useful for expertise in quantum physics. We construct a MATLAB package that aims to simulate the well-known quantum algorithms such as Grover Database Search (GDS) and Quantum Fourier Transform (QFT). This package provides non-experts in quantum physics to play with the simple algorithms to undrestand how quantum algorithms work on an actual quantum computer. The first part of the report deals with the basics of quantum computing, including the definition of ket, bra and various types of quantum gates such as CNOT, SWAP and controlled-U gates. Afterwards, we focus on quantum algorithms for GDS and QFT in detail. Future works will also address the quantum simulation of finite difference modeling needed for seismic wave modeling and inversion. Upcoming part II focuses on the numerical simulation of part I, namely, MATLAB codes for the quantum computational basis generator, random qubit generator with the probability distribution, simulation of GDS and QFT algorithms for general n-qubit states.

INTRODUCTION

Quantum computers potentially can solve the certain class of problems in science much faster than any existing classical computers. In the beginning of 1980s R.P Feynman, a physics Nobel prize winner proposed that if we would be able to run a computer that executes the algorithms quantum mechanically, the running speed up in the simulation of quantum physics would be exponentially faster than classical computers (Feynman, 1982). The first attempt to establish the mathematical framework for quantum computation is proposed by Deutsch (1985). He introduced the concept of the quantum Turing machine called universal quantum computer and also proposed a model for scientific computation based on the quantum physics laws. The first successful algorithm that demonstrated the quantum computation supremacy was introduced by Shor (1999). Given an integer number, Shore algorithm find its prime factors by running in a polynomial time proportional to cubed N. Quantum parallelism exploited for the first time in the Grover search algorithm (Grover, 1996, 1997). This algorithm demonstrates a quadratic speedup for unstructured search on a quantum computer.

Quantum algorithms can be used also to solve the algebraic problems in mathematics and engineering, among them the solution of linear system of the equations which is applicable in many areas in applied science. Given an N-dimensional problem the best classical algorithm offers a computational time proportional to N. However, a quantum computer can solve the linear system of equation in running time log N, a logarithmic speedup compared to the classical computer (Harrow et al., 2009). This novel algorithm, is particularly useful for machine learning (Rebentrost et al., 2014; Lloyd et al., 2013; Wittek, 2014), data

fitting (Wiebe et al., 2012). As the finite difference modeling results in a linear system of the equation, this algorithm can be implemented to solve the wave propagation modeling either by finite difference (Cao et al., 2013) or finite element methods (Montanaro and Pallister, 2016). Generalization of these algorithms to solve seismic wave is an ongoing research and this report is the initial attempt to expand the idea to design and construct the algorithms for seismic problems including modeling, inversion, and imaging. Our attempt in this report is to present an introduction to quantum computing, quantum algorithms and their applications in seismic problems for geophysicists.

The first section concerns basic recalls of quantum theory and addresses the concept of the quantum bit (qubit). We learn the difference between classical bit (cbit) and qubit and the reason behind the magic power of qubit in computation. We also introduce the quantum logic gates and compare them with their classical counterparts. Section 2 some existing quantum algorithms including Deutsch-Joza algorithm, Quantum Database Search (QDS) and Quantum Fourier Transform (QFT). We examine these algorithms in simple cases, for example, QDS for the 3-qubit system and QFT for 2 and 5-qubit system. We illustrate the implementation of QFT in eigenvalue-eigenvector estimation and its role in finite difference modeling. In section 3 we review the most recent quantum algorithms concerning the finite difference modeling. This is where we learn how we can implement the quantum computation to solve the seismic problems. In this section, we illustrate the classical solution for 1,2 and 3D Poisson equation and our approach to solve this problem quantum mechanically. Section 5 provides a MATLAB simulation for quantum computing, including the basic gates and simulation of algorithms in previous sections. We finalize the report by conclusions and future direction.

PERLIMINARIES: QUANTUM MECHANICS, QUBIT AND QUANTUM GATES

A quantum computer is a machine that performs the computation based on the Quantum Mechanics which is the scientific and mathematical description of the behavior of the subatomic particles (Lloyd et al., 1996). A basic difference between the classical and quantum worlds is that a quantum system simultaneously can exist in several physical states. This is one of the principles of QM called superposition principle. This principle is the essence of most of the quantum information and computation tasks. Assume we have a box with a loonie inside. The state of loonie can be either head or tail, so there is a physical reality, no matter you whether open the box to see the coin. It is in one of states, head or tail. However in the quantum regime, based on the superposition principle, before we measure the box, the coin can be in both head and tail states. Right after looking at the box, we force the coin to be in head or tail states. We never see a coin in both head and tail states simultaneously. For more details regarding quantum mechanics and quantum computing, we refer the reader to standard textbooks listed in references (Nielsen and Chuang, 2002; Williams, 2010). In what follows we review the necessary materials to the basic concepts including qubit and elementary quantum gates.

In a classical computation, a single binary variable 0 or 1 is considered as a basic component of the classical computer. The physical realization of the bit can be any physical system exists in either two possible states, for example, two distinct voltages or current levels; two distinct amplitudes of light intensity; two directions of magnetism or polariza-

tion. Qubit is the quantum mechanical version of the bit. It is defined based on the QS principle in a sense that it is in both physical distinct states simultaneously. The mathematical notation that is used to represent a quantum state is ket $|...\rangle$, which for a two-level system is a two-dimensional vector. For example $|0\rangle = (1 \quad 0)^T$ and $|1\rangle = (0 \quad 1)^T$ where T refers to the transpose. Although $|0\rangle$ and $|1\rangle$ are quantum mechanical objects representing the quantum states of the system they are still classical bits. What is called qubit is $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where α and β are the complex amplitudes with probabilistic meaning. $|\alpha|^2$ is the probability of finding the qubit in-state $|0\rangle$ and $|\beta|^2$ is the probability of finding the qubit in-state $|1\rangle$ after we measure the corresponding physical quantity. This quantity can be the spin of a particle or polarization of a photon. We can generalize the 1-qubit register into the n-qubit register. For example two-qubit register is defined as $|\Psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$. Similar to the one-qubit register, the complex coefficients $\alpha_{ij}, ; \ i, j = 0, 1$ are the probability amplitudes. For example, if we measure a property related to this state we get $|00\rangle$ with probability $|\alpha_{00}|^2$ and so on. The two-qubit state is related to a two correlated physical system like two-atom molecule. Similarly, the concept of the qubit can be generalized to the n-qubit register.

The parallel computation in classical regime can be done by having several processors linked together. However, due to the superposition principle in QM, a single quantum processor would be able to perform multiple computing tasks simultaneously. This is called quantum parallelism. In a classical computer logical gates are implemented using the electric current in a circuit including several resistors and transistors. The basic gates are NOT, AND and OR gates. All other gates can be obtained using these three gates. NOT gates is a logical negation, which means it negates the input. In the circuit that explains this gate if the input voltage is zero (value 0) the transistors act as an open switch. As a result, the current flows to the led and it gets on (value 1 for output). Likewise, if we apply the voltage to the input (value 1) the transistor acts as an on key and lets the current flows to the bottom, in this case, led light is off (value 0). OR gate is a logic gate that applies to the two-bit register and the output is the maximum of two inputs. In the circuit, when either voltage applied to input A or B the current flows to the bottom and the led lights up (value 1). Finally, the AND gate, it is a multiplication of the inputs. As we can see from the circuit, the led gets on (output 1) only if the voltage applied to both inputs (A=1 and B=1) (figure1).

In quantum regime, a unitary operator (say U with $UU^\dagger = \mathbb{I}$) acting on a qubit is called quantum logic gate. Since the qubit refers to the state of a quantum system, a quantum gate can be considered as an interaction with the qubit. For example, if we consider the spin state of an atom which can be a superposition of the spin-up and spin-down, by applying the magnetic field we can change the spin of the atom. In this case, the external magnetic field is acting as a single qubit gate. Since a single qubit is a two-dimensional vector, the quantum gate is a 2×2 unitary matrix. As a result, we can define an infinite number of single quantum gates. In contrast, in classical circuit theory for a single bit, there are only two gates, Identity and NOT gates.

Among the single quantum gates, we introduced here the most frequently used. The first one is Hadamard gate. It is defined as $H|0\rangle = 2^{-1/2}(|0\rangle + |1\rangle)$ and $H|1\rangle = 2^{-1/2}(|0\rangle - |1\rangle)$.

NOT(Logical negation)

A (Input)	Y (Output)
0	1
1	0

OR (Maximum between A and B)

A (Input)	B (Input)	Y (Output)
0	0	0
0	1	1
1	0	1
1	1	1

AND (Multiplication of A and B)

A (Input)	B (Input)	Y (Output)
0	0	0
0	1	1
1	0	0
1	1	1

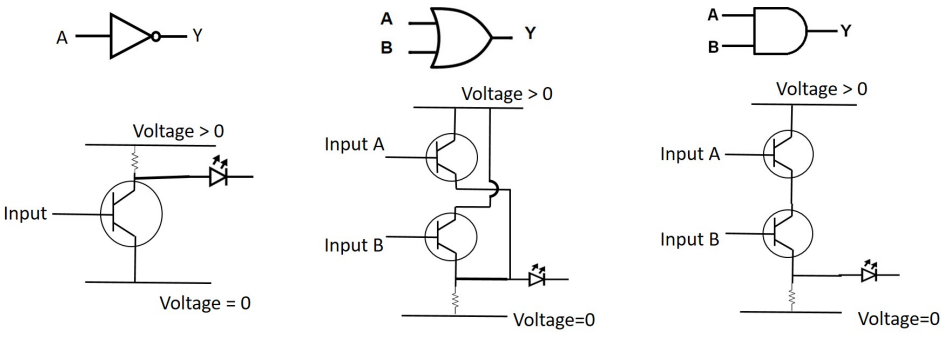

FIG. 1. NOT, OR and AND gates in a classical regime with their physical realizations.

The unitary matrix that corresponds to this gate is

$$H = 2^{-1/2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{1}$$

The action of Hadamard gate on $|x\rangle$ with $x \in \{0, 1\}$ can be written as $H|x\rangle = 2^{-1/2}(|0\rangle + (-1)^x|1\rangle)$, which means the Hadamard gate generate a phase shift between the two generated qubits. Since $H^2 = \mathbb{I}$, the latest equation can be written as $|x\rangle = H\left[2^{-1/2}(|0\rangle + (-1)^x|1\rangle)\right]$. This is an simple but not a concrete example that how we can use Hadamard gate to extract the phase information. Another important single qubit gate is phase shift gate

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & \exp\left(\frac{2\pi i}{2^k}\right) \end{pmatrix} \tag{2}$$

In next section we will use this gate frequently when we describe some simple algorithms. Two qubit gates are unitary 4×4 matrices acting on two qubit states. The most important two qubit gate is CNOT gate which acts as CNOT$|x, y\rangle = |x, x \oplus y\rangle$ where $\oplus$ is the summation modulo 2. For example $0 \oplus 1 = 1$ and $1 \oplus 1 = 0$. For example CNOT$|11\rangle = |10\rangle$ or CNOT$|01\rangle = |01\rangle$. The first qubit x is called controlled qubit and the second one target qubit.

Another gate that is very applicable in quantum algorithms is controlled. Assume S and U are the gates for 1-qubit. The controlled operation is defined as

$$S \oplus U = \begin{pmatrix} S & \mathbf{0} \\ \mathbf{0} & U \end{pmatrix} = \begin{pmatrix} S_{11} & S_{12} & 0 & 0 \\ S_{21} & S_{22} & 0 & 0 \\ 0 & 0 & U_{11} & U_{12} \\ 0 & 0 & U_{21} & U_{22} \end{pmatrix} \tag{3}$$

This gate operates as follows. Operator S(U) applied to the second qubit if the first qubit is $|0\rangle(|1\rangle)$. Let us apply this gate to $|0\rangle|t\rangle$ and $|1\rangle|t\rangle$, where $|t\rangle = \begin{pmatrix} t_1 & t_2 \end{pmatrix}^T$

$$(S \oplus U)|0\rangle|t\rangle \equiv \begin{pmatrix} S_{11} & S_{12} & 0 & 0 \\ S_{21} & S_{22} & 0 & 0 \\ 0 & 0 & U_{11} & U_{12} \\ 0 & 0 & U_{21} & U_{22} \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} S_{11}t_1 + S_{12}t_2 \\ S_{21}t_1 + S_{22}t_2 \\ 0 \\ 0 \end{pmatrix} = |0\rangle S|t\rangle$$

$$(S \oplus U)|1\rangle|t\rangle \equiv \begin{pmatrix} S_{11} & S_{12} & 0 & 0 \\ S_{21} & S_{22} & 0 & 0 \\ 0 & 0 & U_{11} & U_{12} \\ 0 & 0 & U_{21} & U_{22} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ t_1 \\ t_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ U_{11}t_1 + U_{12}t_2 \\ U_{21}t_1 + U_{22}t_2 \end{pmatrix} = |1\rangle U|t\rangle$$

In the case that the first operator is the identity matrix, $(S \oplus U)$ reduces to the controlled U-gate

$$(\mathbb{I} \oplus U)|0\rangle|t\rangle = |0\rangle|t\rangle,$$
$$(\mathbb{I} \oplus U)|1\rangle|t\rangle = |1\rangle U|t\rangle.$$

the generalization of this gate to two qubit as a controlled gate is

$$(S \oplus U \oplus Q \oplus T)|00\rangle|t\rangle \equiv$$
$$\begin{pmatrix} S_{11} & S_{12} & 0 & 0 & 0 & 0 & 0 & 0 \\ S_{21} & S_{22} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & U_{11} & U_{12} & 0 & 0 & 0 & 0 \\ 0 & 0 & U_{21} & U_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & Q_{11} & Q_{12} & 0 & 0 \\ 0 & 0 & 0 & 0 & Q_{21} & Q_{22} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & T_{11} & T_{12} \\ 0 & 0 & 0 & 0 & 0 & 0 & T_{21} & T_{22} \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |00\rangle S|t\rangle.$$

Finally the effect of two qubit controlled gate on the computational basis is

$$(S \oplus U \oplus Q \oplus T)|00\rangle|t\rangle = |00\rangle S|t\rangle,$$
$$(S \oplus U \oplus Q \oplus T)|01\rangle|t\rangle = |01\rangle U|t\rangle,$$
$$(S \oplus U \oplus Q \oplus T)|10\rangle|t\rangle = |10\rangle Q|t\rangle,$$
$$(S \oplus U \oplus Q \oplus T)|11\rangle|t\rangle = |11\rangle T|t\rangle.$$

Another gate which is very useful is SWAP gate. This gate swaps the bits in each basis, for example $\mathrm{SWAP}_{ik}|ijkl\rangle = |kjil\rangle$. For two qubit case SWAP gate operates on $|01\rangle$ as follows

$$\mathrm{SWAP}|01\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = |10\rangle.$$

One of the application of the SWAP gate is up-side down controlled gate,

$$\mathrm{up-dwoncontrolled-U} = \mathrm{SWAP}(\mathbb{I} \oplus U)\mathrm{SWAP}.$$

In this case the first qubit is the target qubit and the second qubit is control qubit

$$\text{SWAP}(\mathbb{I} \oplus U)\text{SWAP}|t\rangle|0\rangle = \text{SWAP}(\mathbb{I} \oplus U)|0\rangle|t\rangle = \text{SWAP}|0\rangle|t\rangle = |t\rangle|0\rangle,$$
$$\text{SWAP}(\mathbb{I} \oplus U)\text{SWAP}|t\rangle|1\rangle = \text{SWAP}(\mathbb{I} \oplus U)|1\rangle|t\rangle = \text{SWAP}|1\rangle U|t\rangle = U|t\rangle|1\rangle.$$

For a comprehensive study of quantum gates we refer readers to Barenco et al. (1995)

QUANTUM ALGORITHMS

Quantum versus classical complexity

Generally, an algorithm defines the steps for performing the complicated algebraic problem. In computer science, the runtime performance is represented by "Big O Notation". In designing algorithms two things are essential. First, the algorithm steps, which means how fast is the algorithm. This is called time complexity. We need to estimate the running time for algorithm before starting the program. Second, the algorithm's memory requirements, how much memory the algorithms needs to solve the problem. This is called space complexity. An efficient algorithm keeps the number of memory registers, which algorithms need small. Both time and space complexities are estimated in terms of the number of n, the size of the input. For an algorithm that executes N times, the complexity is $O(N)$, for example finding a min or max in an unsorted file. In this example, if every step there is another loop that executes N times the complexity increased to $O(N^2)$. Another example of time complexity is $\log(N)$ related to the binary search.

Deutsch's algorithm

Quantum algorithm is a physical process to perform a computational task that utilizes the quantum mechanics. The first and easiest quantum algorithms that display the efficiency of quantum computation over its classical counterpart is Deutsch's algorithm. Here is the statement of the problem. Assume $f(x)$ defined in binary domain $f : \{0,1\} \mapsto \{0,1\}$. There are four possible options for f. $f(0) = f(1) = 0$ or $f(0) = f(1) = 1$ called constant function and $f(0) = 1,\ f(1) = 0$ or $f(0) = 0,\ f(1) = 1$ called balanced function. To determine whether the function is constant or balanced using a classical computer we need to evaluate both $f(0)$ and $f(1)$ which means we need two queries. However, using the Deutsch's quantum algorithm by one query we can determine the type of function f. The algorithm is explained in three steps (figure 2).

- Step 1: we prepare the initial state $|0\rangle|1\rangle$ and apply the Hadamard gate to each qubit
$$|\psi_1\rangle = H|0\rangle H|1\rangle = 2^{-1}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) = 2^{-1}(|0\rangle|0\rangle - |0\rangle|1\rangle + |1\rangle|0\rangle - |1\rangle|1\rangle). \quad (4)$$

- Step 2 in this step we apply the U_f gate to $|\psi_1\rangle$. We know that the effect of this operator on general two qubit state is $U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$. Illustration of the quantum black box U_f is in the Figure 4.
$$\begin{aligned} U_f|\psi_1\rangle =& 2^{-1}U_f(|0\rangle|0\rangle - |0\rangle|1\rangle + |1\rangle|0\rangle - |1\rangle|1\rangle) = \\ & 2^{-1}(U_f|0\rangle|0\rangle - U_f|0\rangle|1\rangle + U_f|1\rangle|0\rangle - U_f|1\rangle|1\rangle) = \\ & 2^{-1}(|0\rangle|0 \oplus f(0)\rangle - |0\rangle|1 \oplus f(0)\rangle + |1\rangle|0 \oplus f(1)\rangle - |1\rangle|1 \oplus f(1)\rangle). \end{aligned}$$

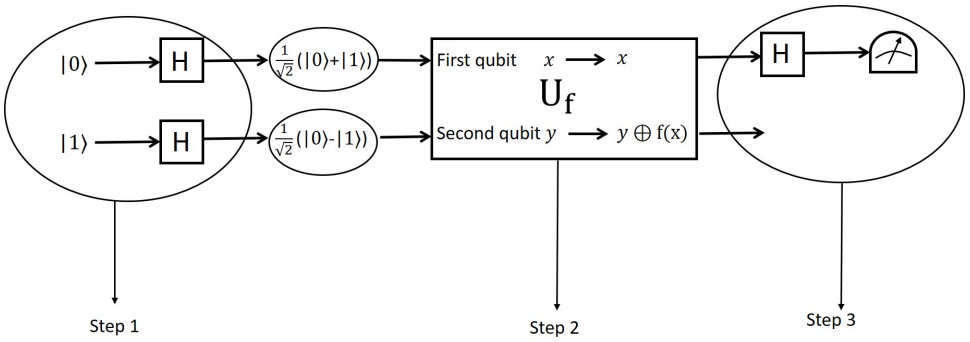

FIG. 2. Illustration of steps for Deustch algorithm.

- Step 3 Finally we apply a Hadamard gate on the first qubit, if the first qubit is $|0\rangle$ function f is constant and if it is $|1\rangle$ function f is balanced.

Let us explain this in more detail. First consider a constant function say $f(0) = f(1) = 0$. In the second step of the algorithm we have

$$\begin{aligned} U_f|\psi_1\rangle =& 2^{-1}(|0\rangle|0\oplus 0\rangle - |0\rangle|1\oplus 0\rangle + |1\rangle|0\oplus 0\rangle - |1\rangle|1\oplus 0\rangle) = \\ & 2^{-1}(|0\rangle|0\rangle - |0\rangle|1\rangle + |1\rangle|0\rangle - |1\rangle|1\rangle) = 2^{-1}\underbrace{(|0\rangle + |1\rangle)}_{\text{first qubit}}(|0\rangle - |1\rangle). \end{aligned}$$

Now in the last step we apply the Hadamard gate to the first qubit

$$HU_f|\psi_1\rangle = 2^{-1}\left\{H(|0\rangle + |1\rangle)\right\}(|0\rangle - |1\rangle) = |0\rangle\left\{2^{-1/2}(|0\rangle + |1\rangle)\right\}.$$

So we find the first qubit in state $|0\rangle$ which means that the function f is constant. Now we examine the algorithm for the balanced function say $f(0) = 1,\ f(1) = 0$ in this case the step 2 reduces to

$$\begin{aligned} U_f|\psi_1\rangle =& 2^{-1}(|0\rangle|0\oplus 1\rangle - |0\rangle|1\oplus 1\rangle + |1\rangle|0\oplus 0\rangle - |1\rangle|1\oplus 0\rangle) = \\ & 2^{-1}(|0\rangle|1\rangle - |0\rangle|0\rangle + |1\rangle|0\rangle - |1\rangle|1\rangle) = -2^{-1}\underbrace{(|0\rangle - |1\rangle)}_{\text{first qubit}}(|0\rangle - |1\rangle). \end{aligned}$$

Now in the last step we apply the Hadamard gate to the first qubit

$$HU_f|\psi_1\rangle = -2^{-1}\left\{H(|0\rangle - |1\rangle)\right\}(|0\rangle - |1\rangle) = -|1\rangle\left\{2^{-1/2}(|0\rangle + |1\rangle)\right\}.$$

As a result, we measure $|1\rangle$ for the first qubit which means that the function f is balanced. So comparing to the classical algorithm, in the quantum case by a single evaluation we can determine whether the function f is constant or balanced. This algorithm can be generalized to the higher dimension. For example evaluation of a function $f(x)$ that can have only values 0 or 1 with x that can have any values from 0 to $2^n - 1$. To determine whether f is constant or balanced with the classical computer we need $2^{n-1}+1$ queries. This algorithm is called Deutsch-Joza algorithm. Without loss of generality lets us explain the algorithm for

$n = 2$, this means x can have values $\{0, 1, 2, 3\}$ which they can assigned to $\{00, 01, 10, 11\}$. An example of constant function in this case is $f(00) = f(01) = f(10) = f(11) = 0$ and an example of balanced function $f(00) = f(01) = 0,\ f(10) = f(11) = 1$. The algorithm is pretty much the same as Deutsch's algorithm except that the input qubit is (n+1)-qubit, with n-zero and one 1 register. This state for case $n = 2$ is $|\psi_0\rangle = |00\rangle|1\rangle$. After applying the Hadamard gate to all qubits we have

$$\begin{aligned}|\psi_1\rangle =& H \otimes H \otimes H \otimes |\psi_0\rangle = H|0\rangle H|0\rangle H|1\rangle = \\ & 2^{-3/2}(|0\rangle + |1\rangle)(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) = \\ & 2^{-3/2}(|000\rangle + |010\rangle + |100\rangle + |110\rangle - |001\rangle - |011\rangle - |101\rangle - |111\rangle).\end{aligned}$$

Now we apply the U_f gate to the above state and remember that it acts on the last qubit

$$\begin{aligned}U_f|\psi_1\rangle =& 2^{-3/2}(|00, 0 \oplus f(00)\rangle + |01, 0 \oplus f(01)\rangle + |10, 0 \oplus f(10)\rangle + \\ & |11, 0 \oplus f(11)\rangle - |00, 1 \oplus f(00)\rangle - |01, 1 \oplus f(01)\rangle - |10, 1 \oplus f(10)\rangle - |11, 1 \oplus f(11)\rangle).\end{aligned}$$

Now assume that the f is constant for example $f(00) = f(01) = f(10) = f(11) = 1$

$$\begin{aligned}U_f|\psi_1\rangle =& 2^{-3/2}(|00, 0 \oplus 1\rangle + |01, 0 \oplus 1\rangle + |10, 0 \oplus 1\rangle + \\ & |11, 0 \oplus 1\rangle - |00, 1 \oplus 1\rangle - |01, 1 \oplus 1\rangle - |10, 1 \oplus 1\rangle - |11, 1 \oplus 1\rangle) = \\ & 2^{-3/2}(|001\rangle + |011\rangle + |101\rangle + |111\rangle - |000\rangle - |010\rangle - |100\rangle - |110\rangle) = \\ & - \left[2^{-1/2}(|0\rangle + |1\rangle)\right] \left[2^{-1/2}(|0\rangle + |1\rangle)\right] \left[2^{-1/2}(|0\rangle - |1\rangle)\right],\end{aligned}$$

finally by applying $H \otimes H \otimes H$ we get

$$(H \otimes H \otimes H)U_f|\psi_1\rangle = -|00\rangle|1\rangle.$$

Now consider to the case where f is balanced say $f(00) = f(01) = 0,\ f(10) = f(11) = 1$

$$\begin{aligned}U_f|\psi_1\rangle =& 2^{-3/2}(|000\rangle + |010\rangle + |101\rangle + |111\rangle - |001\rangle - |010\rangle - |100\rangle - |110\rangle) = \\ & - \left[2^{-1/2}(|0\rangle - |1\rangle)\right] \left[2^{-1/2}(|0\rangle + |1\rangle)\right] \left[2^{-1/2}(|0\rangle - |1\rangle)\right]\end{aligned}$$

finally by applying $H \otimes H \otimes H$ we get

$$(H \otimes H \otimes H)U_f|\psi_1\rangle = -|10\rangle|1\rangle$$

In general after measurement for an initial $(n+1)-$ qubit state if we obtain the final state as $|\underbrace{00000...0}_{\text{n-zero register}}\rangle|1\rangle$, the function f is constant. Other wise, if at least on register in the n-qubit register is 1, the function f is balanced. Comparing to the classical algorithm where we need $2^{n-1} + 1$ queries to evaluate function f, with a quantum algorithm with one query only we can determine whether the function f is constant or balanced.

Quantum database search algorithm (Grover algorithm)

Another important algorithm that displays the quantum speedup over classical counterpart is Grover's search algorithm (Grover, 1996, 1997). This algorithm searches for a specific element among N number of unstructured items. The best classical algorithm solves

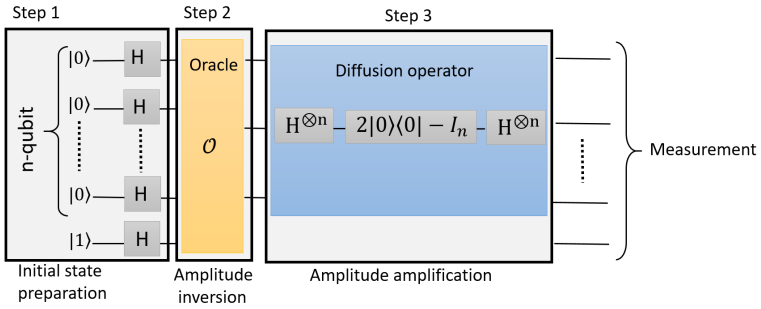

FIG. 3. Illustration of steps for the Grover algorithm.

this problem in $O(N)$ time steps, however, the quantum algorithm gives the quadratic speed up and perform the search in $O(\sqrt{N})$ time step. Just like any other quantum algorithms, Grover's algorithm takes the advantages of superposition principle in which multiple operations on multiple inputs can be performed simultaneously. Let us explain this algorithm step by step. First assume that the search space has $N = 2^n$ items, which means n qubits register can represent the search space items. To utilize the simultaneous operation on N items we need to construct a single state containing N items. Since the search space has N elements, a quantum register with n qubit can store all elements in a single state. To implement this, we start with an initial qubit as

$$|0\rangle^{\otimes n} \equiv \underbrace{|0\rangle \otimes |0\rangle ... \otimes |0\rangle}_{\text{n-times}} = |\underbrace{00000...0}_{\text{n-zero register}}\rangle. \tag{5}$$

Next we apply the Hadamard gate n-times to this initial state

$$|\psi\rangle = H^{\otimes n}|0\rangle^{\otimes n} \equiv \underbrace{H|0\rangle \otimes H|0\rangle ... \otimes H|0\rangle}_{\text{n-times}} =$$

$$\left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right] \otimes \left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right] ... \otimes \left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right] = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle,$$

where

$$x = 0 \equiv \underbrace{00...0}_{n}\,, x = 1 \equiv \underbrace{00..0}_{(n-1)}1, ..., N-1 = 2^n - 1 \equiv \underbrace{11...1}_{n}.$$

The quantum state $|\psi\rangle$ is a superposition of N states. If we measure this state the outcome would be any of this N states with equal probability $1/N$. The next step is to apply a quantum oracle to the state which is a quantum black box by means that the initial state doesn't collapse a classical state, instead, the quantum oracle observed and marked the desired solution. The main point of Grover's algorithm is to amplify the amplitude of the desired state, this is called the amplitude amplification. The whole series of transformation that leads to the solution is called Grover iteration (figure 3). Grover shows that we need $\frac{\pi}{4}\sqrt{N}$ iteration to obtain the optimal probability that the output is the correct answer. Let us explain Quantum Black Box (QBB) in more detail.

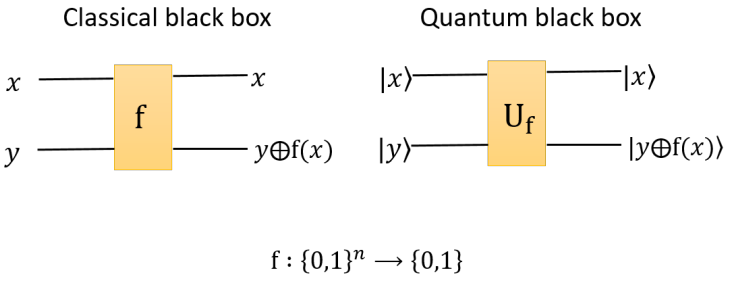

FIG. 4. Classical versus quantum black box.

Assume we have a function $f : \{0,1\}^n \to \{0,1\}$, where $\{0,1\}^n = \{0,1,...,2^n-1\}$. This function takes a n-digit number and the output would be 0 or 1. For example for $n = 3$ function f, can be defined as

$$f = \begin{cases} 0, & x \neq 010, \\ 1, & x = 010, \end{cases} \tag{6}$$

To implement this classical function quantum mechanically, we have to use the QBB (figure 4). An example with application in QDS algorithm is a QBB that negate the amplitude of a specific ket,

$$|x\rangle = \begin{cases} |x\rangle, & x \neq x_f \\ -|x\rangle, & x = x_f, \end{cases} \tag{7}$$

To implement this function we set the target gate as $|y\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. If we apply the QBB we have

$$\begin{aligned} \mathcal{O}_f|x\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) &= \frac{1}{\sqrt{2}}(U_f|x\rangle|0\rangle - U_f|x\rangle|1\rangle) \\ &= \frac{1}{\sqrt{2}}(|x\rangle|0 \oplus f(x)\rangle - |x\rangle|1 \oplus f(x)\rangle). \end{aligned}$$

First assume that $x = x_f$ with $f(x_f) = 1$, then we get

$$\begin{aligned} \mathcal{O}_f|x_f\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) &= \frac{1}{\sqrt{2}}(|x_f\rangle|0 \oplus f(x_f)\rangle - |x_f\rangle|1 \oplus f(x_f)\rangle) \\ &= \frac{1}{\sqrt{2}}|x_f\rangle(|0 \oplus 1\rangle - |1 \oplus 1\rangle) \\ &= \frac{1}{\sqrt{2}}|x_f\rangle(|1\rangle - |0\rangle) = -|x_f\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \end{aligned}$$

Finally after recycling the second qubit we get $|x_f\rangle \to -|x_f\rangle$. In a similar way for $x \neq x_f$ with $f(x \neq x_f) = 0$ we have

$$\begin{aligned} \mathcal{O}_f|\not{x}_f\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) &= \frac{1}{\sqrt{2}}(|\not{x}_f\rangle|0 \oplus f(\not{x}_f)\rangle - |\not{x}_f\rangle|1 \oplus f(\not{x}_f)\rangle) \\ &= \frac{1}{\sqrt{2}}|\not{x}_f\rangle(|0 \oplus 0\rangle - |1 \oplus 0\rangle) \\ &= \frac{1}{\sqrt{2}}|\not{x}_f\rangle(|0\rangle - |1\rangle) = |\not{x}_f\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle), \end{aligned}$$

where $\not{x}_f$ refers to $x \neq x_f$. Finally after recycling the second qubit we get $|\not{x}_f\rangle \rightarrow |\not{x}_f\rangle$. Let us examine the Grover algorithm for $n = 3$ corresponding to $N = 2^3 = 8$ items. Initial state is $|000\rangle$ multiply by ancillary quabit $|1\rangle$. After multiple application of hadamard gates on $|0001\rangle$ we get

$$\begin{aligned}|\Psi_1\rangle &= H^4|0001\rangle = H^3|000\rangle H|1\rangle \\ &= \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\ &- \frac{1}{4}(|0001\rangle + |0011\rangle + |0101\rangle + |0111\rangle + |1001\rangle + |1011\rangle + |1101\rangle + |1111\rangle).\end{aligned}$$

Assume we are looking for $x_f \equiv 011$. Next we apply the oracle to negate the amplitude of the ket associated with x_f

$$\begin{aligned}\mathcal{O}_f|\Psi_1\rangle = \frac{1}{4}(|0000\rangle + |0010\rangle + |0100\rangle - \underline{|0110\rangle} + |1000\rangle + |1010\rangle + |1100\rangle + |1110\rangle) \\ - \frac{1}{4}(|0001\rangle + |0011\rangle + |0101\rangle - \underline{|0111\rangle} + |1001\rangle + |1011\rangle + |1101\rangle + |1111\rangle).\end{aligned}$$

After recycling the ancillary qubit

$$|\mathrm{X}_1\rangle = |\Psi_{000}\rangle - \frac{1}{\sqrt{2}}|011\rangle,$$

where we defined $|\Psi_{000}\rangle$ as

$$|\Psi_{000}\rangle = H^3|000\rangle = \frac{1}{2\sqrt{2}}(|000)\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle).$$

Now we apply the diffusion operator

$$D_1 = 2|\Psi_{000}\rangle\langle\Psi_{000}| - \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \tag{8}$$

to $|\mathrm{X}_1\rangle$, which results

$$D_1|\mathrm{X}_1\rangle = \frac{1}{4\sqrt{2}}(|000)\rangle + |001\rangle + |010\rangle + 5|011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle).$$

We can see at the first iteration we have the amplitude amplification for the desired state $|011\rangle$. In a similar manner after second iteration we arrive at

$$D_3|\mathrm{X}_3\rangle = \frac{1}{8\sqrt{2}}(|000)\rangle + |001\rangle + |010\rangle + 11|011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle).$$

In this step if we measure the state, the probability of finding $|011\rangle$ is

$$\Pr(|011\rangle) = \left|\frac{11}{8\sqrt{2}}\right|^2 = 0.9453. \tag{9}$$

The three major steps of Grover search algorithm illustrated in figure 3.

Quantum Fourier Transform

The Discrete Fourier Transform (DFT) is a very powerful tool in many branches of applied science. It maps an N-dimensional complex vector $\mathbf{X}$ into another N-dimensional complex vector $\mathbf{Y}$

$$\begin{aligned}\text{DFT}:&\mathbf{X}=(x_0,...,x_{N-1})\mapsto\mathbf{Y}=(y_0,...,y_{N-1})\\ &y_k=\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}x_j e^{i2\pi jk/N}.\end{aligned}$$

Quantum Fourier Transform (QFT) is a quantum version of DFT. To assign the quantum state to an N-dimensional complex vector we need only $n=\log_2 N$ qubits. QFT maps the n-qubit state $|x\rangle$ associated to $\mathbf{X}$ into another n-qubit state $|y\rangle$ associated to $\mathbf{Y}$

$$\begin{aligned}\text{QFT}:&|x\rangle=\sum_{j=0}^{N-1}x_j|j\rangle\mapsto|y\rangle=\sum_{k=0}^{N-1}y_k|k\rangle\\ &|j\rangle=\frac{1}{\sqrt{N}}\sum_{k=0}^{N-1}e^{i2\pi jk/N}|k\rangle,\end{aligned}$$

QFT is a $N\times N$ unitary matrix operating on an N-dimensional vector. QFT can perform the DFT efficiently on a quantum computer only with $O(n^2)$ gates however the classical algorithm consists the $O(n2^n)$ gates. To see how QFT algorithm works first let us show how we can estimate the phase of one-qubit state using the application of Hadamard gate. We apply Hadamard gate to a single qubit $|j\rangle$ which can be either $|0\rangle$ or $|1\rangle$

$$\begin{aligned}H|j\rangle&=\frac{1}{\sqrt{2}}(|0\rangle+(-1)^j|1\rangle)=\frac{1}{\sqrt{2}}(|0\rangle+(e^{i\pi})^j|1\rangle)\\ &=\frac{1}{\sqrt{2}}(|0\rangle+e^{ij\pi}|1\rangle)=\frac{1}{\sqrt{2}}(|0\rangle+e^{2i\pi j/2}|1\rangle)\\ &=\frac{1}{\sqrt{2}}(|0\rangle+e^{2i\pi 0.j}|1\rangle).\end{aligned}$$

Now apply the Hadamard gate to both sides one more time, since $H^2=\mathbb{I}$, we get

$$H\frac{1}{\sqrt{2}}(|0\rangle+e^{2i\pi 0.j}|1\rangle)=|j\rangle.$$

As a result of the application of Hadamard gate, we can estimate the phase of a quantum state. This is a simple example of quantum phase estimation. Now assume $|j\rangle$ is a quantum state associated with a fractional number $0<j<1$ which is approximated up to n-digit binary numbers

$$j\equiv 0.j_1j_2j_3...j_n\equiv j_n/2^n+....j_2/2^2+j_1/2^1. \tag{10}$$

The QFT algorithm acts on a n-qubit state $|j\rangle$ as follows

$$\text{QFT}_{2^n}|j\rangle=\frac{1}{\sqrt{2^n}}\left(|0\rangle+e^{2\pi i0.j_n}|1\rangle\right)\left(|0\rangle+e^{2\pi i0.j_{n-1}j_n}|1\rangle\right)...\left(|0\rangle+e^{2\pi i0.j_1j_2...j_n}|1\rangle\right). \tag{11}$$

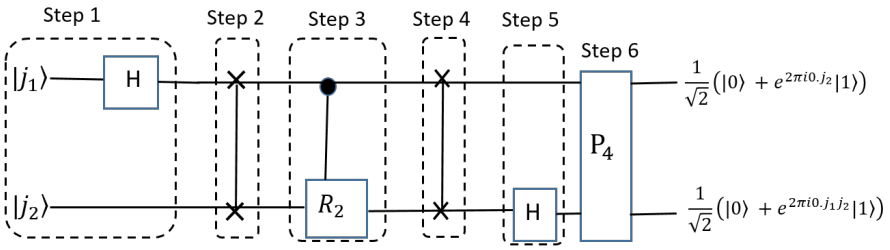

FIG. 5. QFT algorithm step by step for 2-qubit case.

To undrestand how the QFT circuit works, consider the case $n = 2$. In first step Hadamard gate applies to the first qubit

$$H \otimes \mathbb{I}|j_1\rangle|j_2\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.j_1}|1\rangle)|j_2\rangle.$$

In the above operation, H acts on $|j_1\rangle$ and identity matrix $\mathbb{I}$ on $|j_2\rangle$. In second step SWAP gate exchange the first and second qubits

$$\text{SWAP}\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.j_1}|1\rangle)|j_2\rangle = \frac{1}{\sqrt{2}}|j_2\rangle(|0\rangle + e^{2i\pi 0.j_1}|1\rangle).$$

In the next step, a controlled-rotation gate is applied. Let us discuss the effect of $(\mathbb{I} \oplus R_2)$ on a two-qubit state.

$$\begin{aligned}
(\mathbb{I} \oplus R_2)|j_2 = 0\rangle|0\rangle &= |j_2 = 0\rangle|0\rangle \\
(\mathbb{I} \oplus R_2)|j_2 = 1\rangle|0\rangle &= |j_2 = 1\rangle|0\rangle \\
(\mathbb{I} \oplus R_2)|j_2 = 0\rangle|1\rangle &= |j_2 = 0\rangle|1\rangle \\
(\mathbb{I} \oplus R_2)|j_2 = 1\rangle|1\rangle &= e^{2i\pi/4}|j_2 = 1\rangle|1\rangle
\end{aligned}$$

correspondingly we have

$$\begin{aligned}
(\mathbb{I} \oplus R_2)|j_2\rangle|0\rangle &= |j_2\rangle|0\rangle, \\
(\mathbb{I} \oplus R_2)|j_2\rangle|1\rangle &= e^{2i\pi j_2/4}|j_2\rangle|1\rangle.
\end{aligned}$$

and

$$\begin{aligned}
(\mathbb{I} \oplus R_2)\frac{1}{\sqrt{2}}|j_2\rangle(|0\rangle + e^{2i\pi 0.j_1}|1\rangle) &= \frac{1}{\sqrt{2}}\left(|j_2, 0\rangle + e^{2i\pi 0.j_1}e^{2i\pi j_2/4}|j_2, 1\rangle\right). \\
&= \frac{1}{\sqrt{2}}|j_2\rangle(|0\rangle + e^{2i\pi 0.j_1 j_2}|1\rangle).
\end{aligned}$$

In Step 4 we apply the SWAP gate one more time

$$\text{SWAP}\frac{1}{\sqrt{2}}|j_2\rangle(|0\rangle + e^{2i\pi 0.j_1 j_2}|1\rangle) = \frac{1}{\sqrt{2}}\rangle(|0\rangle + e^{2i\pi 0.j_1 j_2}|1\rangle)|j_2\rangle.$$

Then by applying the Hadamard gate we get

$$\frac{1}{\sqrt{2}}(\mathbb{I}\otimes H)(|0\rangle + e^{2i\pi 0.j_1j_2}|1\rangle)|j_2\rangle = \frac{1}{2}(|0\rangle + e^{2i\pi 0.j_1j_2}|1\rangle)H|j_2\rangle,$$
$$= \frac{1}{2}(|0\rangle + e^{2i\pi 0.j_1j_2}|1\rangle)(|0\rangle + e^{2i\pi 0.j_2}|1\rangle)$$

and finally after applying the permutation gate P_4 we reach

$$\mathrm{QFT}_4|j_1\rangle|j_2\rangle = \frac{1}{2}(|0\rangle + e^{2i\pi 0.j_2}|1\rangle)(|0\rangle + e^{2i\pi 0.j_1j_2}|1\rangle). \tag{12}$$

Figure 5 is an illustration of QFT for 2-qubit case.

QUANTUM ALGORITHMS FOR SOLVING LINEAR SYSTEM OF EQUATIONS

Here is the basic idea of the algorithm (Harrow et al., 2009). Assume A is a Hermitian $N \times N$ matrix and $\mathbf{b}$ and $\mathbf{x}$ are the column vectors satisfying $A\mathbf{x} = \mathbf{b}$. The goal is to estimate or find $\mathbf{x} = A^{-1}\mathbf{b}$. Gaussian elimination is a classical algorithm to solve this problem exactly run in time $O(N^3)$. If A is sparse which means no more than $s \ll N$ entries per row, then the conjugate gradient descent method can solve the problem for $\mathbf{x}$ with the computational complexity of $O(N \log N)$. It turns out quantum mechanics can solve this problem better. The quantum mechanical version of the problem is $A|x\rangle = |b\rangle$. The goal is to construct $|x\rangle = A^{-1}|b\rangle$ in a physical process. Since A is hermitian we can assign that to a Hamiltonian for a physical system. First we need $n = \log_2 N$ qubit to encode the information stores in $\mathbf{x}$ into $|x\rangle$, and the same thing for $\mathbf{b}$. Quantum mechanically, A^{-1} acts on $|b\rangle$ through the controlled unitary operator e^{iAt}. Since A is sparse, preparing e^{iAt} takes $O(\log_2 N)$. Now we write the expansion of $|b\rangle$ in terms of eigenvectors of A, $|b\rangle = \sum_j \beta_j|u_j\rangle$. After phase estimation, we obtain, $\sum_j \beta_j|\lambda_j\rangle|u_j\rangle$, where $|\lambda_j\rangle$ is a state that encode the information about eigenvalues λ_j. The next step is to bring the state into $\underline{\sum_j \beta_j\lambda_j^{-1}|u_j\rangle} \otimes |Anc.\rangle$. The underlined part of this state is proportional to $A^{-1}|b\rangle$. The whole algorithm runs in $O(\kappa^2/\epsilon \log N)$, with total error ϵ in the output state. However the best classical algorithm runs in $O(N\sqrt{\kappa}\log \epsilon^{-1})$.

The quantum advantage is when condition numberκ is not too large. We already saw that the output of the algorithm is $|x\rangle = \sum_i x_i|i\rangle$ and the whole process runs in $\log_2 N$, however if we want to read x_i from $|x\rangle$, we still need $O(N)$ effort. As a result the algorithm is useful if we want to measure some features of the solution over some operator M,i.e. expectation value $\langle x|M|x\rangle$. So the complexity is $O(\mathrm{poly}(\log_2 N, \kappa, \epsilon^{-1}))$. In more detail the complexity of each part of the algorithm is as follows. Preparing the quantum state $|b\rangle$ needs $O(\mathrm{poly}(N))$ elementary gates as: $U|\underbrace{00...00}_{N}\rangle \equiv |b\rangle$. For Hamiltonian simulation, it has been shown that for s-sparse Hamiltonian H, the cost is $O(s^2\|Ht\|\mathrm{poly}(\log_2 N, \log_2 \epsilon^{-1}))$ (Berry et al., 2007). Quantum algorithm for linear system of equations has applications in many branches of applied mathematics, such as finite difference and finite element modeling in seismology and computational fluid dynamics.

CLASSICAL ALGORITHMS FOR FINITE DIFFERENCE MODELING: POISSON EQUATION

In this section we review the numerical methods for the solution of the Poisson equation (Demmel, 1997) and possible quantum solutions. Consider the 1-dimensional poisson equation

$$u^{''}(x) = -f(x), \tag{13}$$

with $0 < x < 1$ and boundary condition $u(0) = u(1) = 0$. Figure 6 describes the discretization of 1-D space into $N + 2$ nodes, such that u_i means the value of function u at node x_i, i.e. $u_i \equiv u(x_i)$. The same definition is valid for f. Using the forward (u'_+) and backward (u'_-) derivatives, the second derivative is obtained as

$$u_i^{''} = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \tau_i, \tag{14}$$

where $h = 1/(N + 1)$ and τ_i is the truncation error. Now the Poisson equation reduces to

$$2u_i - u_{i+1} - u_{i-1} = h^2 f_i + h^2 \tau_i. \tag{15}$$

Since in boundaries $u_0 = u_{N+1} = 0$, the above equation is for $0 < i < N + 1$. In matrix form it can be written as

$$\begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & \ddots & & & \\ 0 & -1 & 2 & \ddots & \ddots & & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & 2 & -1 & 0 \\ \vdots & & & \ddots & -1 & 2 & -1 \\ 0 & & & & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix} = h^2 \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix} + h^2 \begin{bmatrix} \tau_1 \\ \tau 2 \\ \vdots \\ \vdots \\ \vdots \\ \tau_{N-1} \\ \tau_N \end{bmatrix} \tag{16}$$

Ignoring the truncation error term τ the above equation reduces to

$$T_N u = h^2 f, \tag{17}$$

where T_N is called Poisson operator which is $N \times N$. It has been shown that using the eigenvectors of Poisson operator we can approximate the solutions of the Poisson equation. Let us calculate the eigenvalues and eigenvectors of T_N. It can be shown that $T_N z_j = \lambda_j z_j$ where

$$\lambda_j = 2\left(1 - \cos\frac{\pi j}{N+1}\right),$$
$$z_j(k) = \sqrt{\frac{2}{N+1}} \sin\left(jk\pi/(N+1)\right).$$

To calculate the condition number we need smallest, λ_{min} and largest, λ_{max}, eigenvalues. For large N

$$\lambda_{max} = \lambda_N = 2\left(1 - \cos\frac{\pi N}{N+1}\right) \approx 4$$
$$\lambda_{min} = 2\left(1 - \cos\frac{\pi j}{N+1}\right) \approx 2\left(1 - \left[\frac{i^2\pi^2}{2(N+1)^2}\right]\right) = \left(\frac{i\pi}{N+1}\right)^2$$

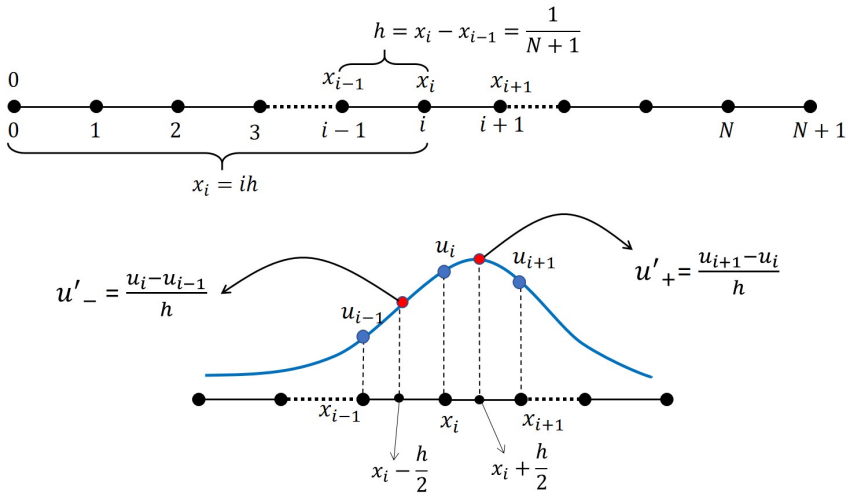

FIG. 6. Finite difference method in one dimension.

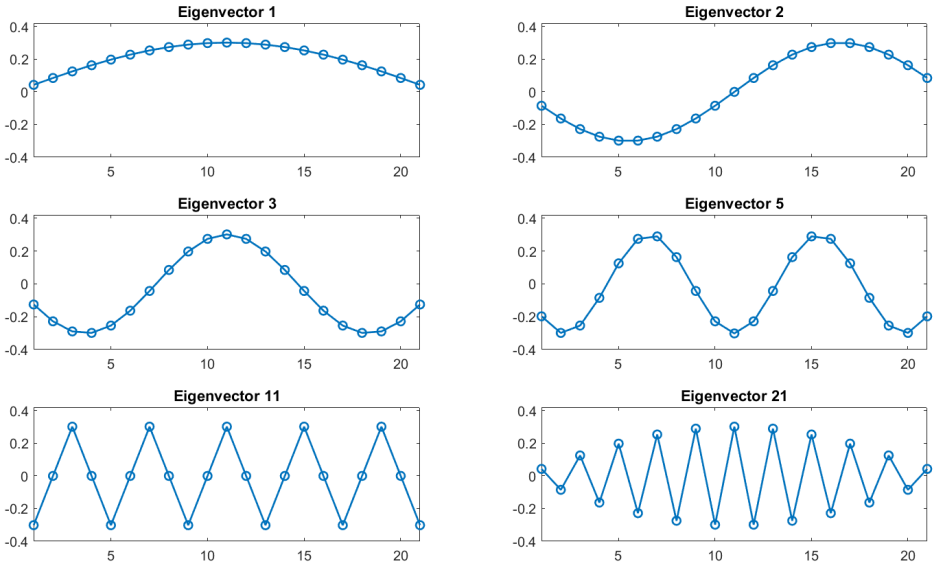

FIG. 7. Six eigenvectors of 1-D poisson operator for $N = 21$

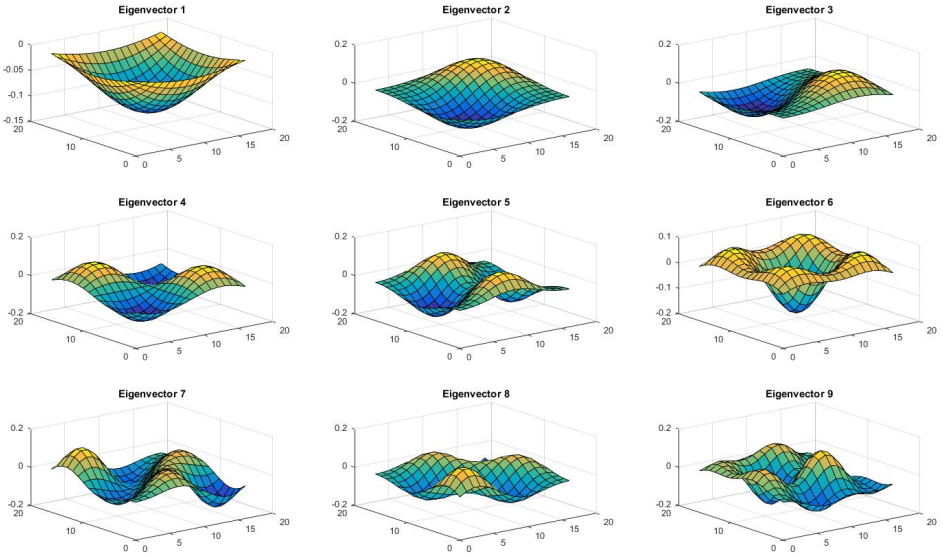

FIG. 8. Nine eigenvectors of 2-D poisson operator for $N = 17$

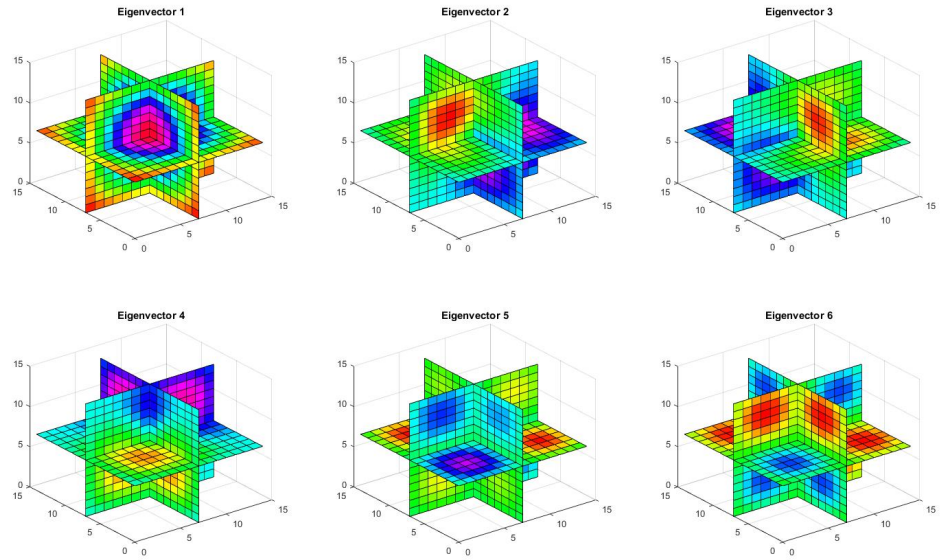

FIG. 9. Six eigenvectors of 3-D poisson operator for $N = 15$

so condition number is $\kappa = \lambda_{max}/\lambda_{min} \approx 4(N+1)^2/\pi^2$. It turns out that eigenvectors of $h^{-2}T_N$ are the approximate eigenvectors of poisson equation. In a similar manner we can write the 2 and 3-D poisson equation matrix as (Demmel, 1997)

$$T_{N\times N} = I_N \otimes T_N + T_N \otimes I_N,$$
$$T_{N\times N\times N} = T_N \otimes I_N \otimes I_N + I_N \otimes T_N \otimes I_N + I_N \otimes I_N \otimes T_N.$$

In figure 7 we plot the six eigenvector of 1-D Poisson equation for $N = 21$. Similar plots for 2-D in figure 8 and for 3-D in figure 9 are shown. As a result, the solution of the Poisson equation reduces to the solution of linear system equation. The best classical algorithm can solve the system of the N linear equation is in order of $O(N)$ using the conjugate gradient method. However, the same problem can be solved roughly by logarithmic speed up using the quantum algorithm only with $O(\log N)$ gates. The strategy to solve this problem quantum mechanically is implementing the QFT. It is well understood that using the QFT we can estimate the eigenvalues and eigenvectors of a unitary operator. Since the eigenvalues of a unitary matrix have modulus one, they can be written as phase $e^{i\phi}$. Therefore to estimate the eigenvalues it is enough to estimate the phase factor. The algorithm that performs this estimation is called Quantum Phase Estimation (QPE). Therefore, the first step to solve the finite difference modeling is to implement the QPE to estimate the eigenvalues and eigenvectors of Poisson operator. After we find out the eigenvector estimation we can employ the appropriate approach and design the quantum algorithm for the next step. How many gates do we need to approximate the solutions? how many registers and ancillary qubits do we need to construct the initial state? these are not known until we figure out the whole algorithm.

NUMERICAL QUANTUM COMPUTING

Computational basis generator: To generate the ket basis for n-qubit system we use the $2^n \times 2^n$ identity matrix. Each column of this matrix gives us a ket. If we start counting the columns from 0 to $2^n - 1$, ket $|k\rangle$ corresponds to the $(k+1)$-th column. For example $|3\rangle = |011\rangle$ is obtained as

```
n=3
I = eye(2^n);
for k=0:2^n-1;
h=dec2bin(k,n);
disp(['|',eval(['num2str(k)']),'>=','|',eval(['num2str(h)']),'>='])
disp([eval(['I(:,k+1)'])])
end
```

```
|0>=|000>=          |1>=|001>=          |2>=|010>=          |3>=|011>=

     1                  0                  0                  0
     0                  1                  0                  0
     0                  0                  1                  0
     0                  0                  0                  1
     0                  0                  0                  0
     0                  0                  0                  0
     0                  0                  0                  0
     0                  0                  0                  0
```

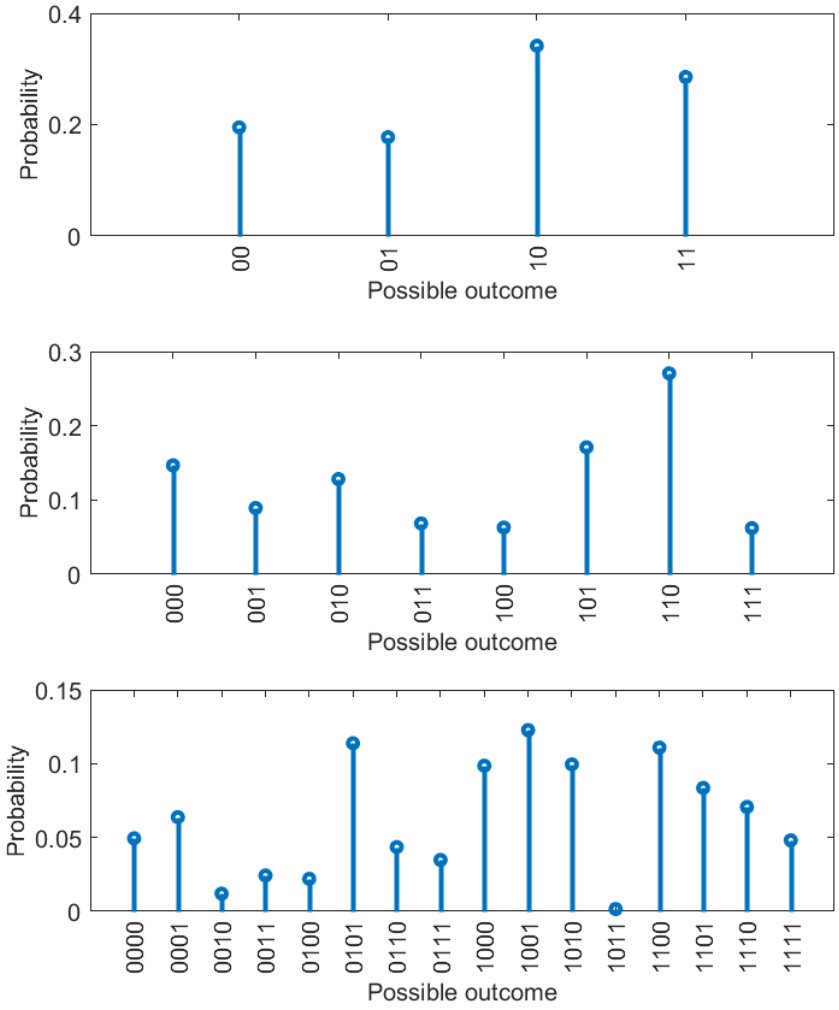

FIG. 10. Probability distributions for possible measurement outcome of a 2,3 and 4-qubit states.

|4>=|100>=	|5>=|101>=	|6>=|110>=	|7>=|111>=
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Random qubit generator: an n-qubit state is a linear superposition of n-basis state. The following MATLAB script generates the random n-qubit state with the probability distribution.

```
n=3;%number of qubits
N=2.^n;%number of registers
a=rand(N,1);%random vector
b=rand(N,1);%random vector
complex=a+1i*b;%complex random vector
qubitN=complex./norm(complex,2)%n-qubit state
prob=abs(qubitN).^2;% probability of each state
stem(prob,'LineWidth',6,'Markersize',15);hold on
xlim([0 N+1]);set(gca,'fontsize',30);
Y=char(blanks(1),dec2bin(0:N-1,n));%qubit x-axis label
set(gca,'XTickLabel',Y);
```

For 2-qubit state the random qubit generated is

```
qubit2 =
   0.3054 + 0.3191i
   0.2735 + 0.3205i
   0.2803 + 0.5130i
   0.1922 + 0.4987i
```

```
qubit3 =
   0.2756 + 0.2662i
   0.1619 + 0.2511i
   0.3471 + 0.0888i
   0.2279 + 0.1288i
   0.1500 + 0.2014i
   0.4016 + 0.0986i
   0.3746 + 0.3611i
   0.2353 + 0.0833i
```

```
qubit4 =
   0.0781 + 0.2083i
   0.0590 + 0.2457i
   0.0787 + 0.0766i
   0.1505 + 0.0406i
   0.1075 + 0.1025i
   0.3190 + 0.1101i
   0.1486 + 0.1465i
   0.0638 + 0.1755i
   0.3126 + 0.0295i
   0.3385 + 0.0907i
   0.1516 + 0.2767i
   0.0384 + 0.0101i
   0.0892 + 0.3209i
   0.1412 + 0.2523i
   0.2055 + 0.1688i
   0.0906 + 0.1999i
```

The probability distribution for 2, 3 and 4 qubit states are given in figure 10.

Initial state preparation: this is the first step in any quantum algorithm. For 3-qubit case we have

```
H=1./(sqrt(2))*[1 1; 1 -1];% hadamard gate
Psi_n=H*[1;0];
for i=1:n-1
Psi_n=kron(H*[1;0],Psi_n);
end
```

```
Psi_n =
   0.3536
   0.3536
   0.3536
   0.3536
   0.3536
   0.3536
   0.3536
   0.3536
```

Quantum gates: let us first consider the controlled gates. In MATLAB we use the `blkdiag` command for direct sum of two operators $S \oplus U$

```
>> blkdiag(S,U)=
```

```
[ S1_1, S1_2,    0,    0]
[ S2_1, S2_2,    0,    0]
[    0,    0, U1_1, U1_2]
[    0,    0, U2_1, U2_2]
```

By application of this operator to a 2-qubit state with $|0\rangle$ and $|1\rangle$ as the first qubit we have

```
blkdiag(S,U)|0>|t>=                    blkdiag(S,U)|1>|t>=

 S1_1*t1 + S1_2*t2                                    0
 S2_1*t1 + S2_2*t2                                    0
                 0                     U1_1*t1 + U1_2*t2
                 0                     U2_1*t1 + U2_2*t2
```

One of the useful two qubit controlled gate is Controlled Not gate. Which in matlab can be generated by `CNOT=blkdiag(I,NOT)`

```
CNOT|00>            CNOT|01>            CNOT|10>            CNOT|11>
   =|00>=              =|01>=              =|11>=              =|10>=
     1                   0                   0                   0
     0                   1                   0                   0
     0                   0                   0                   1
     0                   0                   1                   0
```

For two qubit control gate we have

```
blkdiag(S,U,Q,T)|00>|t>=               blkdiag(S,U,Q,T)|01>|t>=

 S1_1*t1 + S1_2*t2                                    0
 S2_1*t1 + S2_2*t2                                    0
                 0                     U1_1*t1 + U1_2*t2
                 0                     U2_1*t1 + U2_2*t2
                 0                                    0
                 0                                    0
                 0                                    0
                 0                                    0

blkdiag(S,U,Q,T)|10>|t>=               blkdiag(S,U,Q,T)|11>|t>=

                 0                                    0
                 0                                    0
                 0                                    0
                 0                                    0
 Q1_1*t1 + Q1_2*t2                                    0
 Q2_1*t1 + Q2_2*t2                                    0
                 0                     T1_1*t1 + T1_2*t2
                 0                     T2_1*t1 + T2_2*t2
```

Another important gate which is upside-down controlled-U.

```
>> SWAP*blkdiag(eye(2),U)*SWAP
```

```
UpDownContrU=
[ 1,    0, 0,    0]
[ 0, U1_1, 0, U1_2]
[ 0,    0, 1,    0]
[ 0, U2_1, 0, U2_2]
```

For three-qubit case Circuit for Controlled-Controlled-U is defined as a gate that U applied to the third qubit when two first qubit is in state $|1\rangle$.

```
>>blkdiag(I6,U)=
[ 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, U1_1, U1_2]
[ 0, 0, 0, 0, 0, 0, U2_1, U2_2]
```

where `I6` refers to the 6 by 6 identity matrix.

```
blkdiag(I6,U)|110>|t>=                    blkdiag(I6,U)|111>|t>=

    0                                         0
    0                                         0
    0                                         0
    0                                         0
    0                                         0
    0                                         0
 U1_2                                      U1_1
 U2_2                                      U2_1
```

Another gate which is very useful is the SWAP gate. This gate swaps the bits in each qubit, for example, $\texttt{SWAP}_{ik}|ijkl\rangle = |kjil\rangle$. We define all possible `SWAP` operators for n-quabit system as follows

```
>> Row=decimalToBinaryVector(0:N-1)
a=(1:n);
for i=1:n-1;
for j=i+1:n;
anew = a;
idx = a([i j]);
anew(idx) = anew(flip(idx));
eval(['Row',num2str(i),num2str(j),'=Row(:,anew);']);
eval(['indexf',num2str(i),num2str(j),'=binaryVectorToDecimal(Row(:,anew))+1;']);
eval(['SWAP',num2str(i),num2str(j),'=I(:,indexf',num2str(i),num2str(j),')']);
end
end
```

for example the possible `SWAP` operators for $n = 3$ are

```
SWAP12 =

     1     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     0     0     0     1     0     0
     0     0     1     0     0     0     0     0
     0     0     0     1     0     0     0     0
     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     1


SWAP13 =

     1     0     0     0     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     1     0     0     0     0     0
     0     0     0     0     0     0     1     0
     0     1     0     0     0     0     0     0
     0     0     0     0     0     1     0     0
     0     0     0     1     0     0     0     0
     0     0     0     0     0     0     0     1


SWAP23 =

     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     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     1     0
     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     1
```

Permutation gate, for n-qubit system, is defined as

```
Row=decimalToBinaryVector(0:N-1)
Rowflip=Row(:,fliplr(1:n))
idxperm=binaryVectorToDecimal(Rowflip)+1
Perm=I(:,idxperm)
```

Let us explain how this code works. Similar to the SWAP code we first define the Row matrix. Then we flip the rows of the matrix. If we rearrange the columns of the identity matrix according to the new index we obtain the Permutation operator. For example for $n = 2$ and $n = 3$ we have

```
Perm4 =

     1     0     0     0
     0     0     1     0
     0     1     0     0
     0     0     0     1
```

```
Perm8 =

     1     0     0     0     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     1     0     0     0     0     0
     0     0     0     0     0     0     1     0
     0     1     0     0     0     0     0     0
     0     0     0     0     0     1     0     0
     0     0     0     1     0     0     0     0
     0     0     0     0     0     0     0     1
```

Quantum algorithms: A MATLAB code for a quantum oracle is given by

```
for k=0:length(Psi)-1
  Psibits=dec2bin(k,n+1);% assign binary numbers to elemnts of Psi,
  psibits=Psibits(1:end-1);% the binary numbers related to the psi
  lbit=Psibits(end)-'0';% binary related to the last qubit as vector array
  psibits2array=psibits-'0';% binary related to psi as vector array
  psibits2num=bi2de(psibits2array,'left-msb');% conevrt the binary to decimal number
  F(:,k+1)=f(psibits2num+1);% calculate function f for values obtained in last step
  lbitOracle=mod(F(:,k+1)+lbit,2); %calculate |x,a> ----> |x,a+f(x)>
  fidex=[psibits2array lbitOracle];
  fidex_final=bin2dec(num2str(fidex));
  Psinew(k+1,:)=Psi(fidex_final+1); %|x,a+f(x)> ----> (-1)^f(x)|x,a>
end
xf=kron(eye(N),H*[0;1])'*Psinew % dispose the ancilla
```

Let us explain the performance of the quantum oracle on 3-qubit case. In the above code we defined

$$\text{psi} = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)$$

$$\text{Psi} = \text{psi}\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

we also define the function f such as $f(011) = 1$, the above quantum black box find the ket associated with 011 and negate the amplitude. Final result after recycling the ancilla qubit is

$$\text{xf} = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle - |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)$$

Below is the result of the code

```
psi =                       0.3536                  Psi =
                            0.3536                      0.2500
    0.3536                  0.3536                     -0.2500
    0.3536                  0.3536                      0.2500
    0.3536                                             -0.2500
    0.3536                                              0.2500
```

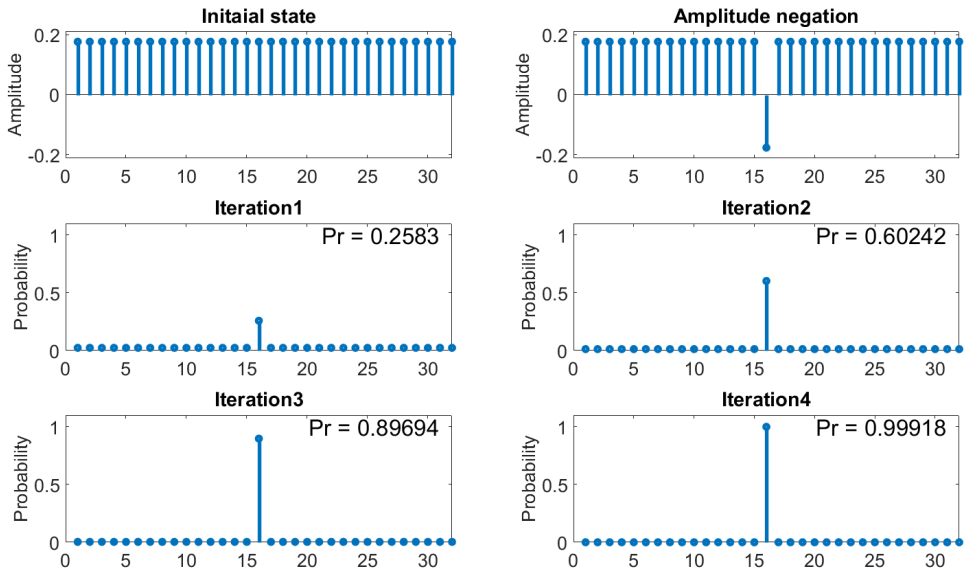

FIG. 11. Initial state preparation and amplitude negation.

```
-0.2500          0.2500          xf=
 0.2500         -0.2500              0.3536
-0.2500                              0.3536
 0.2500                              0.3536
-0.2500                             -0.3536
 0.2500                              0.3536
-0.2500                              0.3536
 0.2500                              0.3536
-0.2500                              0.3536
```

For the Grover search algorithm, in the first step, we construct the initial state by multiple application of the Hadamard gates. In the second step, the QBB negate the amplitude of the marked state corresponding to the correct answer. The final step of the algorithm is called amplitude amplification. In this step, the negated amplitude is increased and the other amplitudes are decreased. Figure11 illustrates the three steps of the algorithms including initial state preparation, amplitude negation and amplitude amplification for search space with 32 entries. Finally the pseudocode that execute the QFT is as follow

for $j = 1 : n - 1$
$\mathbb{I}_{2^{j-1}} \otimes H \otimes \mathbb{I}_{2^{n-j}}$
$S_{j,j+1} \left[\mathbb{I}_{2^{j-1}} \otimes (\mathbb{I} \oplus R_2) \otimes \mathbb{I}_{2^{n-j-1}}\right] S_{j,j+1}$
for $k = 1 : n - j - 1$
$PS_{n-k-j,n-j} \left[\mathbb{I}_{2^{n-j-1}} \otimes (\mathbb{I} \oplus R_{k+2}) \otimes \mathbb{I}_{2^{j-1}}\right] S_{n-k-j,n-j} P$
end
end
$P\mathbb{I}_{2^{n-1}} \otimes H$

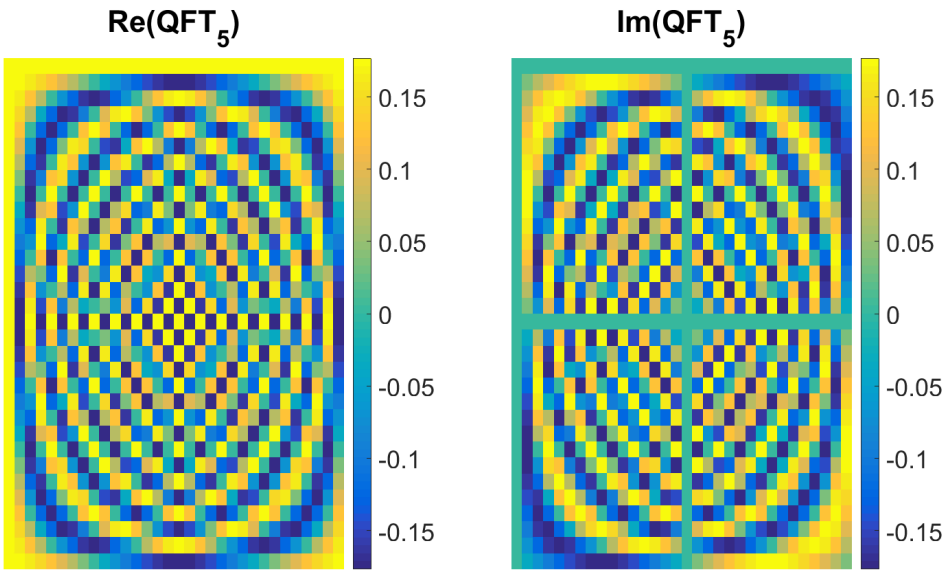

FIG. 12. Quantum Fourier Transform for 5-qubit system.

The result of QFT algorithm for 5-qubit input is shown in figure 12.

DISCUSSION

A short introduction to quantum computing with potential applications to seismic problems is presented in this report. The initial motivation for this report is, if we have a scalable universal quantum computer today, what kind of seismic problems we can solve with it? specifically, what is the quantum algorithm to solve the seismic wave propagation? How can we implement the existing quantum algorithms to construct the new algorithms for modeling, inversion, and migration? One of the most important developed quantum algorithms is QFT that offers an exponential speedup over its classical counterpart. This algorithm is used in eigenvalue-eigenvector estimation which is a key step in the quantum algorithm to solve the linear system of equation (HHL algorithm, Harrow et al. (2009)). Finite difference solution of the seismic wave propagation in frequency domain reduces to the solution of linear system of equation, for example, 1-D wave modeling with N-grids is equal to the solution of N linear equations with N unknown. The best classical algorithm can solve this problem in a time proportional to the number of grids, however, by implementing the HHL algorithm, we can solve the 1D finite difference modeling in log number of grids, i.e. exponential speedup comparing to the classical algorithm. It is known that complexity of finite difference modeling is also proportional to the number of time steps and the number of shots. If a quantum algorithm reduces this complexity to log the number of shots and time steps, this would be the additional speedup. We are currently working on the possible strategy for quantum algorithms to solve the 2D and 3D finite difference modeling. Another possible avenue for further research along these lines is to study the backpropagation of wave field applicable to imaging and migration.

ACKNOWLEDGEMENTS

We thank the sponsors of CREWES for continued support. This work was funded by CREWES industrial sponsors and NSERC (Natural Science and Engineering Research Council of Canada) through the grant CRDPJ 461179-13.

APPENDIX

Axillary qubit recycling

Assume we have two-qubit state $|ab\rangle = |a\rangle \otimes |b\rangle$ and we want to recycle the ket $|a\rangle$ or ket $|b\rangle$. This can be done using the following operation

$$\begin{aligned} &\text{recycling} \quad |a\rangle \qquad \frac{(\langle a| \otimes \mathbb{I})\,|a\rangle \otimes |b\rangle}{\langle a|a\rangle} = |b\rangle, \\ &\text{recycling} \quad |b\rangle \qquad \frac{(\mathbb{I} \otimes \langle b|)\,|a\rangle \otimes |b\rangle}{\langle b|b\rangle} = |a\rangle. \end{aligned}$$

This can be generalized to the higher dimension cases. For example consider to the n-qubit system

$$|a_1 a_2 ... a_j ... a_n\rangle = |a_1\rangle \otimes |a_2\rangle ... |a_j\rangle ... |a_n\rangle.$$

We would like to recycling the $|a_j\rangle$, this can be done as follows

$$|a_1 a_2 ... \cancel{a_j} ... a_n\rangle = \frac{\overbrace{\mathbb{I} \otimes \mathbb{I} ... \mathbb{I}}^{(j-1)-\text{times}} \otimes \langle a_j| \overbrace{\mathbb{I} \otimes \mathbb{I} ... \mathbb{I}}^{(n-j)-\text{times}}}{\langle a_j|a_j\rangle} \otimes |a_1\rangle \otimes |a_2\rangle ... |a_n\rangle.$$

Below is the MATLAB code for qubit recycling

```
a = sym('a', [2 1])% generates the symbolic 2-d column vector a=[a1;a2]
b = sym('b', [2 1])% generates the symbolic 2-d column vector b=[b1;b2]
c = sym('c', [2 1])% generates the symbolic 2-d column vector c=[c1;c2]

norm_a2=norm(a).^2%  norm of vector
norm_b2=norm(b).^2
norm_c2=norm(c).^2

% three qubit state constructed by kroneker product of a, b and c
abc=superkron(a,b,c)

% recovering vector a by recycling a and c
Ibc=superkron(eye(2),b',c');
b_c_recycling=simplify(Ibc*abc./(norm_b2*norm_c2))

% recovering vector b by recycling a and c
aIc=superkron(a',eye(2),c');
a_c_recycling=simplify(aIc*abc./(norm_a2*norm_c2))

% recovering vector c by recycling a and b
abI=superkron(a',b',eye(2));
a_b_recycling=simplify(abI*abc./(norm_a2*norm_b2))
```

REFERENCES

Barenco, A., Bennett, C. H., Cleve, R., DiVincenzo, D. P., Margolus, N., Shor, P., Sleator, T., Smolin, J. A., and Weinfurter, H., 1995, Elementary gates for quantum computation: Physical review A, **52**, No. 5, 3457.

Berry, D. W., Ahokas, G., Cleve, R., and Sanders, B. C., 2007, Efficient quantum algorithms for simulating sparse hamiltonians: Communications in Mathematical Physics, **270**, No. 2, 359–371.

Cao, Y., Papageorgiou, A., Petras, I., Traub, J., and Kais, S., 2013, Quantum algorithm and circuit design solving the poisson equation: New Journal of Physics, **15**, No. 1, 013,021.

Demmel, J. W., 1997, Applied Numerical Linear Algebra: SIAM.

Deutsch, D., 1985, Quantum theory, the church-turing principle and the universal quantum computer, *in* Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, vol. 400, The Royal Society, 97–117.

Feynman, R. P., 1982, Simulating physics with computers: International journal of theoretical physics, **21**, No. 6, 467–488.

Grover, L. K., 1996, A fast quantum mechanical algorithm for database search, *in* Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, ACM, 212–219.

Grover, L. K., 1997, Quantum mechanics helps in searching for a needle in a haystack: Physical review letters, **79**, No. 2, 325.

Harrow, A. W., Hassidim, A., and Lloyd, S., 2009, Quantum algorithm for linear systems of equations: Physical review letters, **103**, No. 15, 150,502.

Lloyd, S., Mohseni, M., and Rebentrost, P., 2013, Quantum algorithms for supervised and unsupervised machine learning: arXiv preprint arXiv:1307.0411.

Lloyd, S. et al., 1996, Universal quantum simulators: SCIENCE-NEW YORK THEN WASHINGTON-, 1073–1077.

Montanaro, A., and Pallister, S., 2016, Quantum algorithms and the finite element method: Physical Review A, **93**, No. 3, 032,324.

Nielsen, M. A., and Chuang, I., 2002, Quantum computation and quantum information.

Rebentrost, P., Mohseni, M., and Lloyd, S., 2014, Quantum support vector machine for big data classification: Physical review letters, **113**, No. 13, 130,503.

Shor, P. W., 1999, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer: SIAM review, **41**, No. 2, 303–332.

Wiebe, N., Braun, D., and Lloyd, S., 2012, Quantum algorithm for data fitting: Physical review letters, **109**, No. 5, 050,505.

Williams, C. P., 2010, Explorations in quantum computing: Springer Science & Business Media.

Wittek, P., 2014, Quantum machine learning: what quantum computing means to data mining: Academic Press.