

Introduction to Qiskit

Instructor: Prof Subhamoy Maitra

Author: Subha Kar, MTech Crs2021, ISI Kolkata

February 13, 2022

Introduction

In this paper I explain the basic components of the quantum paradigm so that one can start writing programs on a suitable platform. We describe several basic features of quantum bits (qubits) and related operations such as quantum gates, and explain how they operate. We also try to explain how working on a quantum platform provides . In this regard, teleportation, super-dense coding and some of well known quantum algorithm are discussed. I support the discussion with codes written in Qiskit SDK.

Programming a quantum computer is now something that anyone can do in the comfort of their own home. But what to create? What is a quantum program anyway? In fact, what is a quantum computer?

Initialization

“IBM Quantum Services provides a secure, portable, containerized runtime and programming tools to access the latest world-leading cloud-based quantum systems and simulators via the IBM Cloud.” Detailed information regarding this can be found at <https://www.ibm.com/quantum-computing/>. The programming SDK Qiskit is available at <https://qiskit.org/>—all the examples here are written using Qiskit . As the web pages are quite self-explanatory, get into the programming using the toolkit. In Qiskit, a qubit is initialized in the following manner.

```
1 #Importing the required classes and modules
2 from qiskit import QuantumRegister
3
4 qr = QuantumRegister(1, "q")
```

The above code will initialize a single qubit to $|0\rangle$ and name the qubit as q (naming is optional). Let us break the code to understand better. Line 1 is simply a comment. In line 2, we import the QuantumRegister class which is a

necessary step to use the class later. In line 4, we create a QuantumRegister class object and assign it to the variable qr. While instantiating the class, we provide the number of qubits and the name of the qubit as the parameters. Here, we have given the number of qubits as 1 and the name of the qubit as q. Apart from $|0\rangle$, the state of the qubit qr can also be set to any valid arbitrary state. This is done either by evolving the qubit qr using gates (which we will cover in the next section) or using the initialize() method. For instance, the qubit qr can be set to the quantum state $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ using initialize() as in the following code.

```






1 #Importing the required classes and modules
2 from qiskit import QuantumRegister, QuantumCircuit
3 from math import sqrt
4
5 desired_state = [1/sqrt(2), 1/sqrt(2)]
6 qr = QuantumRegister(1, "q")
7 qc = QuantumCircuit(qr)
8 qc.initialize(desired_state, qr[0])

```

Let us analyze the above code by line. In line 2, we import the QuantumRegister and QuantumCircuit classes. The code in the following line is used to import the “square root” mathematical function. We then assign desired state the vector $[\alpha, \beta]$, where $\alpha|0\rangle + \beta|1\rangle$ is the state of the qubit we desire to create. Next, we create a QuantumRegister object qr. Line 7 of the code creates a QuantumCircuit object qc with qr as the qubit of the circuit. We then use the initialize() method to initialize the qubit qr to the state specified by desired state.

Single Qubit Gates

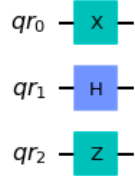
Quantum gates are operations that evolve the state of qubit(s). Any gate operation should transform a valid state of a qubit to another valid state of the same. An important characteristic of a quantum gate is that any quantum gate operation is reversible. That is if a quantum gate acts on an input state $|\psi\rangle$ give some output state $|\psi\rangle$, then given the state $|\psi\rangle$, we should be able to obtain the state $|\psi\rangle$ through some valid quantum gate operation. Vectorially, a gate operation is a transformation from a normalized complex vector to another. These norm preserving operations are performed using unitary transformations. Any unitary transformation can be represented by a unitary matrix. So, corresponding to any quantum gate, there exists a unitary matrix. Some well known single qubit gate are **X, H, Y, Z** etc. Their matrix representation are given below.

Operator	Gate(s)		Matrix
Pauli-X (X)			$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)			$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)			$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)			$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

Let us now see how we implement the gates in Qiskit. The syntax for the above mentioned gates are as follows

```
[9]: qr=QuantumRegister(3,'qr')
circuit=QuantumCircuit(qr)
circuit.x(qr[0])
circuit.h(qr[1])
circuit.z(qr[2])
circuit.draw()
```

[9]:



Single Qubit Measurement

Measurement is the process of observing the state of a qubit. Measurement is also a qubit operation. However, unlike the single qubit gates, measurement is not a unitary operation, and not even reversible. Measuring a qubit gives one bit of information. The peculiarity of measurement is that observing a qubit “collapses”, i.e, changes the state of the qubit to one of the basis states. For instance, if we measure a qubit of the form $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, we obtain $|0\rangle$ with probability $\frac{1}{2}$ and $|1\rangle$ with probability $\frac{1}{2}$. The $\frac{1}{2}$ is obtained by squaring the absolute value of the amplitude of the respective states.

The following code applies an **H** gate on the qubit and measures the qubit in Computational basis.

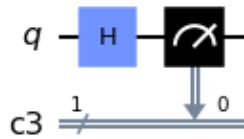
```

from qiskit import *
qr=QuantumRegister(1,'q')
cr=ClassicalRegister(1)
qcircuit=QuantumCircuit(qr,cr)
qcircuit.h(qr)
qcircuit.measure(qr,cr)
backend=Aer.get_backend('qasm_simulator')
qjob=execute(qcircuit,backend,shots=25)
counts=qjob.result().get_counts()
print ( counts )
qcircuit.draw()

```

```
{'1': 9, '0': 16}
```

```
:
```



Multiple Qubits

Even though single qubits are instrumental in understanding the bedrock of quantum computing, if we want to actually solve problems of considerable scale, we cannot aim to do that without multiple qubits. A multi-qubit system can be thought of as an extension to the single qubit system. Let us first take a simple scenario of two qubits. If we have two classical bits, then all possible states of the two bit system are 00, 01, 10 and 11. In a quantum setting, similar to a single qubit, a two qubit system can be in a superposition of all four possible states. In Dirac's notation, the two qubit system can be written as $|\psi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle$, such that sum of α_i square is 1.

Multi-Qubit Gates

Consider a system of two qubits. We want to perform a NOT operation on the second qubit only if the state of the first qubit is $|1\rangle$. To achieve this in a classical setting, we just perform an xor of the first and the second bits and store it in the second bit. It is obvious that we cannot perform this operation using single qubit gates because the single qubit gates can only manipulate a single qubit and is independent of the other qubits. So, we need a gate that can interact with two qubits at the same time, in other words, a two qubit gate. To perform the controlled operation mentioned above, we use a CNOT gate, also

called a “Controlled-NOT” gate. The CNOT gate has two inputs, a ‘control’ qubit and a ‘target’ qubit. If the control qubit is set to $|1\rangle$ then it flips the target qubit and does nothing otherwise. The matrix representation of the CNOT gate is given by the following 4×4 unitary matrix.

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

The action of **CNOT** on a two qubit state of the form $|\psi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle$ can be given as **CNOT** $|\psi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|11\rangle + \alpha_3|10\rangle$.

The syntax for applying a CNOT in a quantum circuit qc is
qc . cx (q0 , q2)

Entanglement

Entanglement is another property of qubits that differentiate them from the classical bits. Consider the following two qubit state

$$|\psi\rangle = \alpha|00\rangle + \beta|11\rangle$$

The state $|\psi\rangle$ above cannot be written as a tensor product of two single qubit states. The state of any one of the two qubits alone cannot be determined in a precise manner. In this case, the two qubits are said to be in an entanglement.

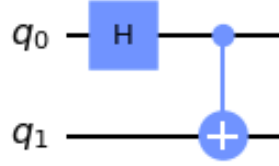
Creating an Entanglement

Qiskit program for cracting Entangle state is as follows

```
[15]: from qiskit import *
qr=QuantumRegister(2,'q')
qcircuit=QuantumCircuit(qr)
# Apply H-gate to the first:
qcircuit.h(qr[0])
# Apply a CNOT:
qcircuit.cx(qr[0],qr[1])

qcircuit.draw()
```

[15]:



we use the command “from qiskit import *”. This implies that all the necessary objects that are embedded in Qiskit will be imported. Here, we can see that we are considering a quantum register with two qubits and a classical register with two bits using the initializations $q = \text{QuantumRegister}(2, 'q')$. This is because we are now dealing with two qubits. The two qubits are placed in an array. First qubit is denoted by $q[0]$ whereas the second qubit is denoted by $q[1]$. As for the creation of the entanglement, in the quantum circuit qc , the Hadamard gate (h) is applied on $q[0]$ followed by CNOT with $q[0]$ as the control and $q[1]$ as the target qubits, respectively. The evolution of the states take place as follows.

$$|0\rangle |0\rangle \xrightarrow{H \otimes I} \frac{1}{\sqrt{2}} \left(|0\rangle + |1\rangle \right) |0\rangle \xrightarrow{CNOT_{0,1}} \frac{1}{\sqrt{2}} \left(|0\rangle |0\rangle + |1\rangle |1\rangle \right).$$