Quantum Information Science

Miguel Sozinho Ramalho

November, 2018

Table of contents

Introduction

This week we are going to build the bridge between **Quantum Mechanics** and **Quantum Computing**, in other words, how the Quantum laws can be leveraged into the tools we need for, well, computing! This will be done through a parallelism with Classical Computing.
Furthermore, we will also go through our first **Qiskit snippets** and finish of by running our first Quantum circuit on a **real Quantum Processor**, using IBM Q Experience. Qasm language will also be briefly mentioned as a tool for this course.

Quantum Information Science

Is a broad area of studies, it is the Quantum sibling of typical information science. It is concerned with **representing**, **manipulating** and **maintaining** information in quantum states. It tackles many problems that did not exist, at least in the same form, in Classical Computing, such as *quantum error correction*, *quantum teleportation*, *quantum communication*, ...

Classical bits

The simplest unit of information in classical systems, a bit (short for binary unit), which can store either a 0 or a 1 value, thus binary.

There is quite an interesting metaphor for explaining the difference between bits and their quantum equivalent, based on the concept of coin tossing, from which we will start!

Coin tossing for bits

When you toss a coin, the result will either be tails or heads - 0 or 1 (please try not to think of the cases of a vertical landing until you can do it). This is a binary and deterministic state.

Tails = 0

Heads = 1

classical register: is an array of n independent bits. A 1-bit register is simply a bit and can be in 1 out of $2^1 = 2$ possible states, whereas an 8-bit register can be in 1 out of $2^8 = 256$ states.

qubit

Assuming you have a quantum state (isolated from interference) that has not been measured. If we refer back to the 2 possible states from Week 0, we know our state is a combination of both 0 and 1, remember?

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

But what does this mean, exactly? Well, that our system is not in just one of the states (assuming $\alpha \neq 0 \wedge \beta \neq 0$), it holds the information of both possible states, at the same time.

Coin tossing for qubits

And so, instead of heads or tails, we can compare this state to a coin that is still spinning. Such is the essence of the qubit, a simultaneous state of 0 and 1 (described according to the probability distribution of α and β). Notice that the state is not hidden according to the probabilities, but rather comprised of both possibilities!

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

qubit

Unlike a bit, we now have a single quantum state with two simultaneous states. Why does this matter? Because this uncertainty contains in itself much more than a deterministic bit, so that when we perform operations, they are applied to all possible states and not just the one.

Quantum register: is an array of n qubits. A 1-bit quantum register is simply a qubit and can hold $2^1 = 2$ possible states, whereas an 8-qubit quantum register can hold $2^8 = 256$ states, not just 1 of them like classical registers, all of them!!

Coin tossing metaphor

If we have one coin, the state **can be** 0 or 1.

...

Coin tossing metaphor

If we have two coins,the state **can be** 00 or 01 or 10 or 11.

...

Coin tossing metaphor

If we have three coins,the state **can be** 000 or 001 or 010 or 011 or 100 or 101 or 110 or 111.

...

Coin tossing metaphor

Essentially if we have **n** coins, we can have 1 of the 2^n possible states.
For n=4 ($2^4 = 16$ possible states), 1010 would be:

Coin tossing metaphor

If we have one spinning coin, the state **is** 0 and 1.

...

Coin tossing metaphor

If we have two (independent) spinning coins, the state **is** 00 and 01 and 10 and 11.

...

Coin tossing metaphor

If we have three (independent) spinning coins, the state **is** 000 and 001 and 010 and 011 and 100 and 101 and 110 and 111.

...

Coin tossing metaphor

Essentially if we have **n** (independent) spinning coins, we can have 2^n possible states simultaneously.
For $n = 4$ our state holds $2^4 = 16$ possibilities. The information we can have grows **exponentially** with the number of spinning coins or, let me unveil the curtain, qubits! Such is the power of the qubit, and this "supercharged" version of the bit will help us understand why Quantum Computing really tips the scales.

Curiosity*

As the bits also have higher order units (trit for a ternary state, ...) so does the qubit have its d-order equivalent: the **qudit** (quantum d-git).
For the initial case of the hydrogen atom, we could simply consider it as having 3 possible orbits, thus $|0\rangle$, $|1\rangle$ and $|2\rangle$ (a qudit with $d = 3$ is actually a qutrit - quantum trit).
Nevertheless, we will not spend much time with these units as their use is not so straightforward, and once you master qubits, it is easier to extrapolate to other arities than the other way around!

Hands-on - Registers

Let us now write some python that will follow us through many lessons to come.
Here's how to create a ClassicalRegister on Qiskit:

```
from qiskit import ClassicalRegister
# Create a Classical Register with 2 bits.
c = ClassicalRegister(2)
```

Likewise for QuantumRegister:

```
from qiskit import QuantumRegister
# Create a Quantum Register with 2 qubits.
q = QuantumRegister(2)
```

For our purpose, classical registers will serve only to save the results of measurements on qubits.

Hands-on - Quantum Circuit

To connect our classical and quantum registers in a QuantumCircuit we do:

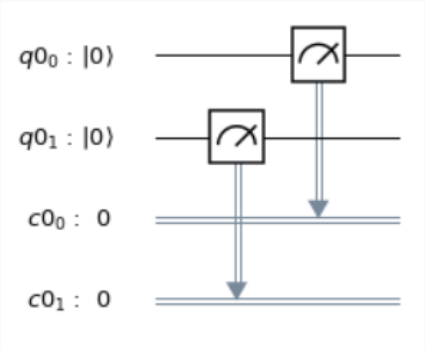
```
from qiskit import QuantumCircuit
# Create a Quantum Circuit
qc = QuantumCircuit(q, c)
# perform a measurement of our qubits into our bits
qc.measure(q, c)
```

What we can do so far is quite limited, but these are the building blocks we need. In the next lesson we will take a look at the operations that can happen before we measure a quantum circuit!

Hands-on - Quantum Circuit Visualization

Here is the code for visualizing our mighty complex circuit:

```
from qiskit.tools.visualization
 import matplotlib_circuit_drawer as draw
draw(qc) # visualize our quantum circuit
```



You will notice that both qubits ($q0_0$ and $q0_1$) are initially in state $|0\rangle$ meaning that $\beta = 0$ in $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ (what do you think $\alpha =$?). This is by design and how most experiments begin. The symbol connecting the qubit to each bit is the universal symbol for quantum measurement.

Qasm

Qasm derives from 'Open Quantum Assembly Language' and reads 'kazm'. This is a rather recent invention, coming of of a 2017 paper. It is a descriptive language that maps a quantum circuit as text instructions. It has since became a standard and, although we will not be going deeper into it, it is good to understand the overall structure of these documents, here is the example for the above circuit (q0 was changed into q, and c0 to c due to some Qasm interpreters):

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c[2];
x q[0];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

QASM + IBM Q Experience exercise

IBM Q Experience supports QASM in their online editor, and you can literally use a GUI and check the Qasm equivalent (and vice versa). Your task is to go to the editor and do the following:

- Login into IBM Q Experience
- Click on "New" for a new experiment
- Name it as you like (eg. "qasm_test")
- Choose ibmqx2 or ibmqx4 (look for available)
- Click on "Switch to Qasm editor"
- Paste the above Qasm code and see the visual result
- Press "Simulate" and see the result (should be 00 with 1.000 frequency, this means that out of all the repetitions of the experiment, 100% resulted in 00).
- Go ahead and press "Run" and, just for this once, ignore if there is a cached version and choose "New Execution"!

You just executed instructions on a Quantum Computer, you will receive an email with the results (may be more queued jobs ahead of you).

Qiskit exercise

The code provided here has been written to a Jupyter Notebook that you are encouraged to execute on your machine, as that exercise will help you understand Qiskit from the beginning. There is also a code sample for generating the Qasm instructions and, at the end of the notebook, there are more suggested exercises. This, along with the previous slide on testing IBM Q Experience are your tasks for the week. Feel free, of course, to take some extra steps and trying out new stuff on your version of the notebook!

Where to learn more?

- Qasm documentation
- General article on Hackernoon to widen your view
- Veritaserum video on qubits
- Eye-opening and funny commic on Quantum Computing