

Study Notes:

Quantum Computing in Slow Pace

量子計算慢慢來

2021 (CC BY-NC-SA 4.0) Elton Huang

Draft Version

尚未校稿・僅供參考

持續更新。歡迎回饋、意見或提問

Email: eltonhuang@yahoo.com

LINE id: [jesusinelson](#)



October 31, 2021

Contents

<i>What is this?</i>	3
1 Quantum Gates	4
1.1 Basics from Physics to Math	4
1.1.1 Bloch Sphere	5
1.1.2 Products of Matrices	7
1.1.3 Quantum Annealing	7
1.2 Part I	8
1.2.1 Hadamard	8
1.2.2 CNOT	9
1.2.3 Bell States	13
1.2.4 Teleportation	14
1.3 Part II	16
1.3.1 SWAP	16
1.3.2 Toffoli (AND/OR/XOR)	17
2 Quantum Algorithms, Exercises and Advanced Topics	18
2.1 Deutsch-Jozsa	18
2.1.1 Deutsch's Algorithm with 1 Qubit	18
... Implementation with Qiskit	20
2.1.2 Deutsch-Josza Algorithm with 2 Qubits	26
... 3-Qubit Matrix Work	31
... Implementation with Qiskit	39
2.2 More Oracle-Based	46
2.2.1 Bernstein-Vazirani	46
2.2.2 Simon	46
2.3 Grover's	46
2.4 Exercise: Superposition	47
2.5 Quantum Circuit Synthesis	48
3 Applications	49
3.1 Optimization	49
3.1.1 Quadratic Problems with Qiskit	49

3.1.2	Knapsack with Ocean	49
3.2	Monte Carlo Method	49
3.3	Chemical Simulation	50
3.3.1	Molecular Dynamics with Qiskit	50
3.3.2	Published Works	50
3.3.3	OpenFermion	50
3.4	Machine Learning	51
3.4.1	TensorFlow Quantum	51
3.4.2	Qiskit ML	51
4	Explore and Exercise	52
4.1	IBM Quantum <i>Errors</i>	52
4.2	Virtual Quantum Optics Laboratory	56
	Appendices	57
A	ezqc.py	58
B	qmtx.py	61

What is this?

This document is merely a collection of notes during my course studying Quantum Computing. Most of the introductory texts in Quantum Computing do not include extensive elaborations of the matrix derivations involved in developing a concept perhaps due to the large print spans they would take. My notes started with these exercise works when I *built my intuition* to understand the subject matter.

Later it expanded to serve to structure the topics of interests as well as notes and thoughts on various fundamental and application topics, and for sharing too. Related works of Python codes are also included.

Chapter 1

Quantum Gates

1.1 Basics from Physics to Math

- For a layperson introduction of Quantum Theory: *The Fabric of the Cosmos: Quantum Leap* (PBS NOVA, Nov 17, 2011).
- Dr. Shankar at Yale gave a fairly clear narrative on Quantum Mechanics for a term in 2011 ¹
 - Thomas Young’s Double Slit Experiment, 2013
 - Fourier Transform
 - Euler’s Equation: $e^{ix} = \cos(x) + i \sin(x)$

Wave-Particle Duality of Matter; Schrödinger Equation (MIT OCW 2017, 光電效應實驗), but as for 40:02:

- Feynman’s Derivation of the Schrödinger Equation
- How did Schrödinger end up with his equation? How did he decide that his wave was going to be scalar as opposed to a vector field like E&M wave? Etc.

Schrodinger’s Equation

Introduction to Quantum Mechanics: Schrodinger Equation

Nobody knows how the wave function collapses:

- Copenhagen: 由於觀測而發生塌縮 (量子論縱覽, 人人出版2020, Page 79)
- Many World: decoherence of quantum state when interact with the environment which effectively “split” the universe into mutually unobservable alternate histories. (More on Wikipedia)

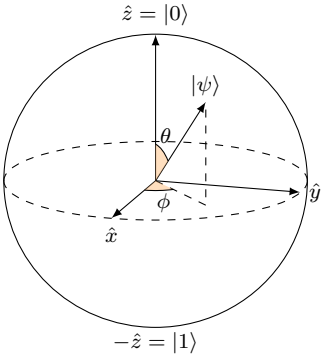
“Researchers observed that **carbon atoms** can tunnel. They thus overcome an energetic barrier, although they do not actually possess enough energy to do that. ... At very low temperatures under ten Kelvin, one molecule form is significantly preferred due to the energy difference.”

Carbon displays quantum effects, Ruhr-Universitaet-Bochum, 2017

¹Compton Effect

1.1.1 Bloch Sphere

“These representations on the Bloch’s circle (sphere) are combinations of the two wave functions. ... What is the physical meaning of the circle (sphere)? ... There is no physical meaning. The Bloch’s circle (sphere) is a visualization; it’s a way thinking about these vectors. ...”, Cameron Akkar.

$\frac{-\hbar^2}{2m} \frac{d^2\psi}{dr^2} + V\psi = E\psi$		$\begin{aligned} \psi\rangle &= \cos\left(\frac{\theta}{2}\right) 0\rangle + e^{i\phi} \sin\left(\frac{\theta}{2}\right) 1\rangle \\ &\equiv e^{i\phi_\alpha} \cos\left(\frac{\theta}{2}\right) 0\rangle + e^{i\phi_\beta} \sin\left(\frac{\theta}{2}\right) 1\rangle, \\ \phi &= \phi_\beta - \phi_\alpha \\ &= \alpha 0\rangle + \beta 1\rangle \end{aligned}$	$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$
--	---	---	---

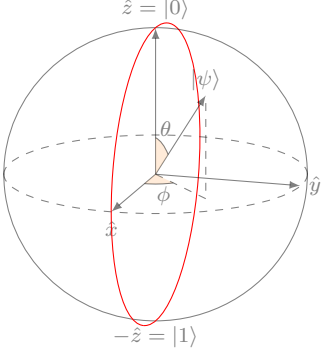
$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} : \theta = 0^\circ, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} : \theta = 180^\circ.$$

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

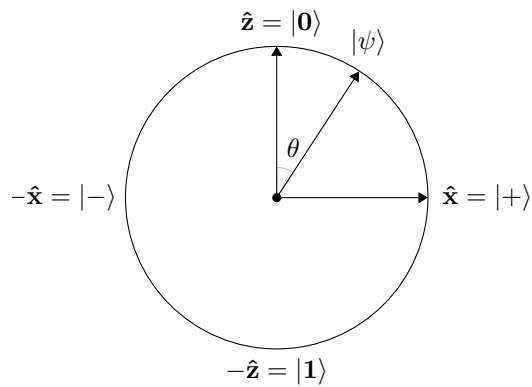
$$Z|1\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} = -|1\rangle, \quad \theta = -180^\circ.$$

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

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

$\frac{-\hbar^2}{2m} \frac{d^2\psi}{dr^2} + V\psi = E\psi$		$ \psi\rangle = \alpha 0\rangle + \beta 1\rangle$	$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$
--	---	---	---

For the sake of simplicity, if we maintain $\phi = 0^\circ$,²



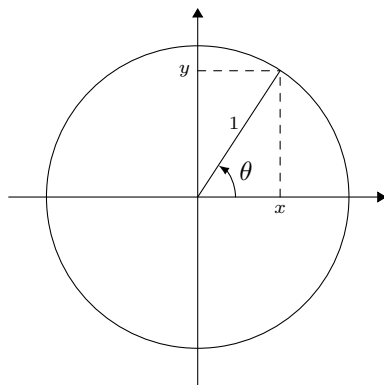
$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + \sin\left(\frac{\theta}{2}\right)|1\rangle$$

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

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

§8.5 *Limitations of the quantum operations formalism* in *Quantum Computation and Quantum Information* (Nielson and Chuang 2010, Page 394 bottom)

Compare:



$$\cos(\theta) = x$$

$$\sin(\theta) = y$$

²Okay up till the Grover's

1.1.2 Products of Matrices

Inner & outer products — Lecture 5 — Matrix Algebra for Engineers

Tensor Product

1.1.3 Quantum Annealing

Webinar: Quantum Computing by a Quantum Annealer

Quantum annealing with more than one hundred qubits

D-Wave

1.2 Part I

1.2.1 Hadamard

$$|0\rangle + |1\rangle + |2\rangle + |3\rangle \longrightarrow \boxed{f(x) = 2x^2 - 5x + 6} \longrightarrow |6\rangle + |3\rangle + |4\rangle + |9\rangle$$

參考：鍾豪著，零與一之間的威力/量子電腦的原理，科學月刊 2021 (5 月號) 617 期，第 26 頁
Quantum Computing as a High School Module (Perry et al., 2020, Page 71 §9.2 Limitations)

注意：不能只單純理解為 qubit 進入 $|0\rangle$ 和 $|1\rangle$ 機率各半的狀態

$$|0\rangle \longrightarrow \boxed{H} \longrightarrow \boxed{H} \longrightarrow |0\rangle$$

$$|1\rangle \longrightarrow \boxed{H} \longrightarrow \boxed{H} \longrightarrow |1\rangle$$

還是要知道在 Bloch's Sphere 或是利用(複數)矩陣運算上的定義。

參考 *Quantum Computing as a High School Module* (Perry et al., 2020, Page 50 top)

1.2.2 CNOT

學習單

先完成 (A), (B), (C), (D) : (還沒學過矩陣乘法的同學可以參考連結)

如果 $q_0 = |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ 而 $q_1 = |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$,

$$(A) \ q_1 q_0 = |10\rangle = |?\rangle \otimes |?\rangle = \begin{pmatrix} ? \times \begin{pmatrix} ? \\ ? \end{pmatrix} \\ ? \times \begin{pmatrix} ? \\ ? \end{pmatrix} \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix}, \text{ 參考連結}$$

也就是 $a_{00} = ?, a_{01} = ?, a_{10} = ?, a_{11} = ?$. 若讓 q_1 以 CNOT 控制 q_0 ,

$$CNOT|10\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix}, \text{ 也就是 } |??\rangle \text{ (如下)}$$

同樣地，

$$(B) \ |00\rangle = |?\rangle \otimes |?\rangle = \begin{pmatrix} ? \times \begin{pmatrix} ? \\ ? \end{pmatrix} \\ ? \times \begin{pmatrix} ? \\ ? \end{pmatrix} \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix}$$

$$CNOT|00\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix} = |??\rangle$$

$$(C) \ |01\rangle = |?\rangle \otimes |?\rangle = \begin{pmatrix} ? \times \begin{pmatrix} ? \\ ? \end{pmatrix} \\ ? \times \begin{pmatrix} ? \\ ? \end{pmatrix} \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix}$$

$$CNOT|01\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix} = |??\rangle$$

$$\begin{aligned}
(\text{D}) \quad |11\rangle &= |?\rangle \otimes |?\rangle = \begin{pmatrix} ? \times \begin{pmatrix} ? \\ ? \end{pmatrix} \\ ? \times \begin{pmatrix} ? \\ ? \end{pmatrix} \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix} \\
CNOT|11\rangle &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix} = |??\rangle
\end{aligned}$$

但是請注意： $a_{00}, a_{01}, a_{10}, a_{11}$ 可能是滿足 $a_{00}^2 + a_{01}^2 + a_{10}^2 + a_{11}^2 = 1$ 的任何實數組合
各 qubit 可能處於不同的疊加態。例如，若

$$q_0 = |\alpha\rangle = \frac{\sqrt{3}}{2} |0\rangle + ? |1\rangle = \frac{\sqrt{3}}{2} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + ? \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{3}}{2} \\ ? \end{pmatrix},$$

$$q_1 = |\beta\rangle = \frac{3}{5} |0\rangle + ? |1\rangle = \frac{3}{5} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + ? \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \end{pmatrix},$$

$$q_1 q_0 = |\beta\alpha\rangle = |\beta\rangle \otimes |\alpha\rangle = \begin{pmatrix} ? \times \begin{pmatrix} ? \\ ? \end{pmatrix} \\ ? \times \begin{pmatrix} ? \\ ? \end{pmatrix} \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix}, \text{ (注意這四個數的平方和為1)}$$

$$CNOT|\beta\alpha\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ ? \end{pmatrix} = ?|00\rangle + ?|01\rangle + ?|10\rangle + ?|11\rangle$$

Solution

If $q_0 = |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $q_1 = |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$,

$$q_1 q_0 = |10\rangle = |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 1 \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$a_{00} = 0, a_{01} = 0, a_{10} = 1, a_{11} = 0$. Let q_1 control q_0 with CNOT,

$$CNOT|10\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \text{ i.e. } |11\rangle \text{ by below}$$

likewise,

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 0 \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$CNOT|00\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |00\rangle$$

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 1 \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 0 \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$CNOT|01\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = |01\rangle$$

$$|11\rangle = |1\rangle \otimes |1\rangle = \begin{pmatrix} 0 \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 1 \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$CNOT|11\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = |10\rangle$$

But please note: $a_{00}, a_{01}, a_{10}, a_{11}$ may be any combination of real numbers that satisfies $a_{00}^2 + a_{01}^2 + a_{10}^2 + a_{11}^2 = 1$. (Each qubit may be in different variance of superposition of states.) For example, if

$$q_0 = |\alpha\rangle = \frac{\sqrt{3}}{2} |0\rangle + \frac{1}{2} |1\rangle = \frac{\sqrt{3}}{2} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \frac{1}{2} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{pmatrix},$$

$$q_1 = |\beta\rangle = \frac{3}{5} |0\rangle + \frac{4}{5} |1\rangle = \frac{3}{5} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \frac{4}{5} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{3}{5} \\ \frac{4}{5} \end{pmatrix},$$

$$q_1 q_0 = |\beta\alpha\rangle = |\beta\rangle \otimes |\alpha\rangle = \begin{pmatrix} \frac{3}{5} \times \begin{pmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{pmatrix} \\ \frac{4}{5} \times \begin{pmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \frac{3\sqrt{3}}{10} \\ \frac{3}{10} \\ \frac{4\sqrt{3}}{10} \\ \frac{4}{10} \end{pmatrix}, \text{ (note that the squares of all elements sum up to 1)}$$

$$CNOT |\beta\alpha\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{3\sqrt{3}}{10} \\ \frac{3}{10} \\ \frac{4\sqrt{3}}{10} \\ \frac{4}{10} \end{pmatrix} = \begin{pmatrix} \frac{3\sqrt{3}}{10} \\ \frac{3}{10} \\ \frac{4}{10} \\ \frac{4\sqrt{3}}{10} \end{pmatrix} = \frac{3\sqrt{3}}{10} |00\rangle + \frac{3}{10} |01\rangle + \frac{4}{10} |10\rangle + \frac{4\sqrt{3}}{10} |11\rangle$$

The 2 qubits are entangled now, so we can't decompose them into 2 separated qubits for distinct wave probabilities for each qubit as

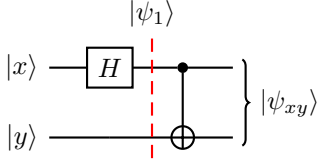
$$\begin{pmatrix} \frac{3\sqrt{3}}{10} \\ \frac{3}{10} \\ \frac{4}{10} \\ \frac{4\sqrt{3}}{10} \end{pmatrix} = |\delta\gamma\rangle, \text{ If } |\gamma\rangle = c_0 |0\rangle + c_1 |1\rangle, \text{ and } |\delta\rangle = d_0 |0\rangle + d_1 |1\rangle,$$

$$|\delta\gamma\rangle = |\delta\rangle \otimes |\gamma\rangle = \begin{pmatrix} d_0 \times \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} \\ d_1 \times \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} d_0 c_0 \\ d_0 c_1 \\ d_1 c_0 \\ d_1 c_1 \end{pmatrix} = \begin{pmatrix} \frac{3\sqrt{3}}{10} \\ \frac{3}{10} \\ \frac{4}{10} \\ \frac{4\sqrt{3}}{10} \end{pmatrix}$$

Xref: *Quantum Computing as a High School Module* (Perry et al., 2020, Page 58 §7.4 second example).

The No-Cloning Theorem: you can't treat the qubits in entanglement individually.

1.2.3 Bell States



Let $|x\rangle = x_0 |0\rangle + x_1 |1\rangle = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$, $|y\rangle = y_0 |0\rangle + y_1 |1\rangle = \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}$

$$H|x\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} x_0 + x_1 \\ x_0 - x_1 \end{pmatrix}$$

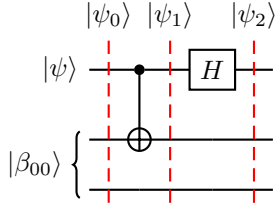
$$|\psi_1\rangle = (H|x\rangle) \otimes |y\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} (x_0 + x_1)y_0 \\ (x_0 + x_1)y_1 \\ (x_0 - x_1)y_0 \\ (x_0 - x_1)y_1 \end{pmatrix}$$

$$|\psi_{xy}\rangle = CNOT|\psi_1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} (x_0 + x_1)y_0 \\ (x_0 + x_1)y_1 \\ (x_0 - x_1)y_0 \\ (x_0 - x_1)y_1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} (x_0 + x_1)y_0 \\ (x_0 + x_1)y_1 \\ (x_0 - x_1)y_1 \\ (x_0 - x_1)y_0 \end{pmatrix}$$

$ \psi_{xy}\rangle$	$ y\rangle = 0\rangle$ $y_0 = 1, y_1 = 0$	$ y\rangle = 1\rangle$ $y_0 = 0, y_1 = 1$
$ x\rangle = 0\rangle$ $x_0 = 1$ $x_1 = 0$	$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \left(\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right) = \frac{ 00\rangle + 11\rangle}{\sqrt{2}}$	$\frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \left(\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right) = \frac{ 01\rangle + 10\rangle}{\sqrt{2}}$
$ x\rangle = 1\rangle$ $x_0 = 0$ $x_1 = 1$	$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{2}} \left(\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right) = \frac{ 00\rangle - 11\rangle}{\sqrt{2}}$	$\frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ -1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \left(\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right) = \frac{ 01\rangle - 10\rangle}{\sqrt{2}}$

Xref: *Quantum Computation and Quantum Information* (Nielson and Chuang 2010, Page 26)

1.2.4 Teleportation



Let $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$

$$|\psi_0\rangle = |\psi\rangle \otimes |\beta_{00}\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} \alpha \times \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \\ \beta \times \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} \alpha \\ 0 \\ 0 \\ \alpha \\ \beta \end{pmatrix} = \frac{1}{\sqrt{2}}(\alpha(|000\rangle + |011\rangle) + \beta(|100\rangle + |111\rangle)), \text{ (exercise: verify)}$$

$$|\psi_1\rangle = (CNOT_{2qb} \otimes I_{1qb}) |\psi_0\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} |\psi_0\rangle = \begin{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & 0 & 0 & 0 \\ 0 & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & 0 & 0 \\ 0 & 0 & 0 & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ 0 & 0 & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & 0 \end{pmatrix} |\psi_0\rangle$$

$$= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ 0 \\ 0 \\ \alpha \\ \beta \\ 0 \\ 0 \\ \beta \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} \alpha \\ 0 \\ 0 \\ \alpha \\ \beta \\ 0 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}}(\alpha(|000\rangle + |011\rangle) + \beta(|101\rangle + |110\rangle))$$

$$|\psi_2\rangle = (H_{1qb} \otimes I_{1qb} \otimes I_{1qb}) |\psi_1\rangle = (\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}) |\psi_1\rangle$$

$$= (\frac{1}{\sqrt{2}} \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right) \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}) |\psi_1\rangle = (\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}) |\psi_1\rangle$$

$$= \frac{1}{2} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} \alpha \\ 0 \\ 0 \\ \alpha \\ 0 \\ \beta \\ \beta \\ 0 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} \alpha \\ \beta \\ \beta \\ \alpha \\ \alpha \\ -\beta \\ -\beta \\ \alpha \end{pmatrix}$$

$$= \frac{1}{2} (\alpha(|000\rangle + |011\rangle + |100\rangle + |111\rangle) + \beta(|001\rangle + |010\rangle - |101\rangle - |110\rangle))$$

$$= \frac{1}{2} (|00\rangle (\alpha|0\rangle + \beta|1\rangle) + |01\rangle (\alpha|1\rangle + \beta|0\rangle) + |10\rangle (\alpha|0\rangle - \beta|1\rangle) + |11\rangle (\alpha|1\rangle - \beta|0\rangle))$$

1.3 Part II

1.3.1 SWAP

Let $|\alpha\rangle = a_0|0\rangle + a_1|1\rangle = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$, $|\beta\rangle = b_0|0\rangle + b_1|1\rangle = \begin{pmatrix} b_0 \\ b_1 \end{pmatrix}$

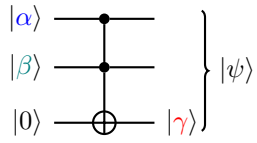
SWAP $|\alpha\beta\rangle = |\beta\alpha\rangle$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_0b_0 \\ a_0b_1 \\ a_1b_0 \\ a_1b_1 \end{pmatrix} = \begin{pmatrix} a_0b_0 \\ a_1b_0 \\ a_0b_1 \\ a_1b_1 \end{pmatrix} = \begin{pmatrix} b_0a_0 \\ b_0a_1 \\ b_1a_0 \\ b_1a_1 \end{pmatrix}$$

1.3.2 Toffoli (AND/OR/XOR)

Xref: *Quantum Computing for Everyone*, Bernhardt 2019, Chapter 6, Page 90~91

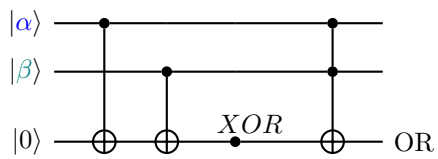
AND



$$|\psi\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_0\beta_0 \\ 0 \\ \alpha_0\beta_1 \\ 0 \\ \alpha_1\beta_0 \\ 0 \\ \alpha_1\beta_1 \\ 0 \end{pmatrix} = \begin{pmatrix} \alpha_0\beta_0 \\ 0 \\ \alpha_0\beta_1 \\ 0 \\ \alpha_1\beta_0 \\ 0 \\ 0 \\ \alpha_1\beta_1 \end{pmatrix} = \alpha_0\beta_0 |000\rangle + \alpha_0\beta_1 |010\rangle + \alpha_1\beta_0 |100\rangle + \alpha_1\beta_1 |111\rangle$$

$ \psi\rangle$	$ \beta\rangle = 0\rangle$ $\beta_0 = 1, \beta_1 = 0$	$ \beta\rangle = 1\rangle$ $\beta_0 = 0, \beta_1 = 1$
$ \alpha\rangle = 0\rangle$ $\alpha_0 = 1$ $\alpha_1 = 0$	$ 000\rangle$	$ 010\rangle$
$ \alpha\rangle = 1\rangle$ $\alpha_0 = 0$ $\alpha_1 = 1$	$ 100\rangle$	$ 111\rangle$

OR/XOR



Chapter 2

Quantum Algorithms, Exercises and Advanced Topics

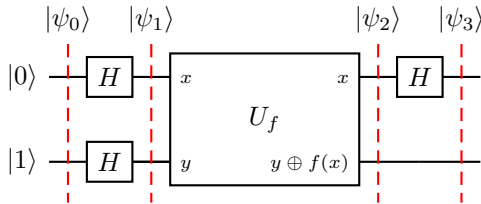
2.1 Deutsch-Jozsa

2.1.1 Deutsch's Algorithm with 1 Qubit

The function $f(x)$ is:

“constant” if regardless the input x , $f(x)$ always produces the same results; i.e. $f(|0\rangle) = f(|1\rangle)$.

“balanced” if for different input x , $f(x)$ produces different results; so $f(|0\rangle) \neq f(|1\rangle)$.



The Deutsch's algorithm:

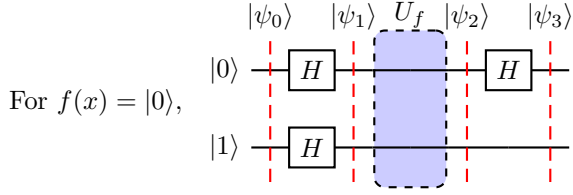
1. Design the oracle U_f according to $f(x)$ so that $U_f |x, y\rangle = |x, y \oplus f(x)\rangle$.

It can be one of the four circuits below.

2. By looking at the state of 1st qubit of output of the circuit $(H \otimes I)U_f(H \otimes H)(|0\rangle \otimes |1\rangle)$ (i.e. $|\psi_3\rangle$ in the diagram above), can determine whether $f(x)$ is constant or balanced. ¹

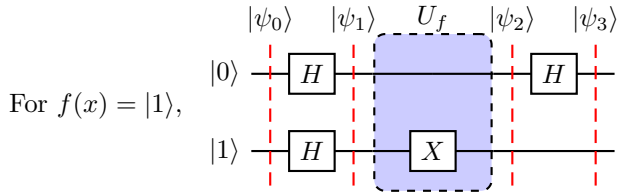
$$|\psi_1\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} = \frac{|00\rangle - |01\rangle + |10\rangle - |11\rangle}{2} = \frac{1}{2} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}$$

¹ $|\alpha\beta\rangle = |\alpha\rangle |\beta\rangle = |\alpha\rangle \otimes |\beta\rangle$



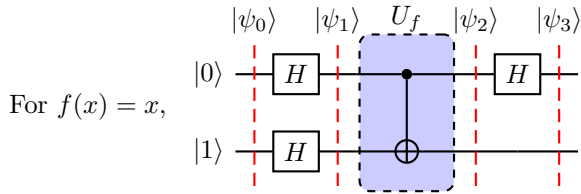
$$|\psi_2\rangle = |\psi_1\rangle$$

$$|\psi_3\rangle = \frac{1}{2\sqrt{2}} \begin{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & -\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \end{pmatrix} = \frac{|00\rangle - |01\rangle}{\sqrt{2}} = |0\rangle \otimes |-\rangle$$



$$|\psi_2\rangle = (I \otimes X) |\psi_1\rangle = \frac{1}{2} \begin{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & 0 \\ 0 & \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} -1 \\ 1 \\ -1 \\ 1 \end{pmatrix}$$

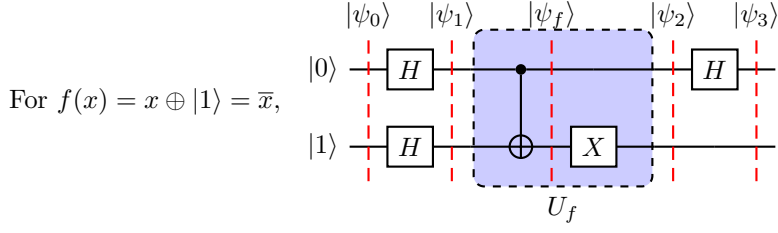
$$|\psi_3\rangle = -\frac{|00\rangle - |01\rangle}{\sqrt{2}} = |0\rangle \otimes (-|-\rangle)^2$$



$$|\psi_2\rangle = CNOT |\psi_1\rangle = \frac{1}{2} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix}$$

$$|\psi_3\rangle = \frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 0 \\ 1 \\ -1 \end{pmatrix} = \frac{|10\rangle - |11\rangle}{\sqrt{2}} = |1\rangle \otimes |-\rangle$$

² $|q\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle$: with $-|q\rangle$, $\frac{\theta}{2}$ turns 180° , i.e. θ turns 360° .



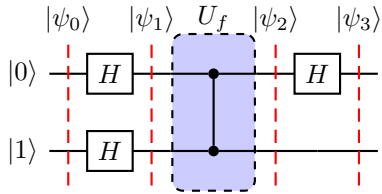
$$|\psi_2\rangle = (I \otimes X) |\psi_f\rangle = \frac{1}{2} \begin{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & 0 \\ 0 & \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} -1 \\ 1 \\ 1 \\ -1 \end{pmatrix}$$

$$|\psi_3\rangle = |1\rangle \otimes (-|-\rangle)$$

Observation:

In order to reflect the balanced nature of the function, the oracle brings the data and target qubits into entanglement whereas the oracles for the constant functions do not, thus with the oracles for constant functions, the second Hadamard gate brings the qubit back to it's original state $|0\rangle$.

The CZ Oracle



$$|\psi_2\rangle = CZ |\psi_1\rangle = \frac{1}{2} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ -1 \\ 1 \\ 1 \end{pmatrix}$$

$$|\psi_3\rangle = \frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ -1 \end{pmatrix} = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$

... Implementation of 1-Qubit Deutsch's Algorithm with Qiskit (Next Page)

djqskt

October 10, 2021

Qiskit 實作：Deutsch-Josza
2021 (CC BY-NC-SA 4.0) Elton Huang

```
[1]: from ezqc import *
      from math import pi

      # 定義 the Deutsch's Algorithm

      def dj1 (uf):
          qc = EzQC (2)
          qc.set_qubits([(0, 0), (1, 1)], bb = False)
          qc.h (0)
          qc.h (1)
          qc.barrier ()
          uf (qc)
          qc.barrier ()
          qc.h (0)
          qc.measure (0, 0)
          qc.draw("mpl")
          return ('constant' if list (run_real (qc, shots = 1).keys())[0][1] == '0'
          ↪ else 'balanced')
          # 理論上只跑一次就夠了，會回傳：{'0x': 1} (results of lower qubit indices,
          ↪ top of circuit plots, on the right)
          # q1 is of value ket-0 due to no measurement (Todo: verify)
```

uf1c00: $f(x) = |0\rangle$, constant
uf1c01: $f(x) = |0\rangle$, constant
uf1c10: $f(x) = |1\rangle$, constant
uf1b00: $f(x) = x$, balanced
uf1b10: $f(x) = x \oplus |1\rangle = \bar{x}$, balanced

```
[2]: # 定義 1 qubit 的 4 種 Uf (oracles)

      def uf1c00 (qc): # constant
          pass

      def uf1c01 (qc): # constant
```

```

qc.p (pi, 1)

def uf1c10 (qc): # constant
    qc.x (1)

def uf1b00 (qc): # balanced
    qc.cnot (0, 1)

# def uf1b01 (qc): # balanced?
#     qc.cz (0, 1)

def uf1b10 (qc): # balanced
    qc.cnot (0, 1)
    qc.x (1)

```

```

[4]: from random import shuffle

# 將 4 個 Uf 存到一個 list 中，並註記名稱
uf1s = [ ('uf1c00', uf1c00), ('uf1c10', uf1c10), ('uf1b00', uf1b00), ('uf1b10', uf1b10),
        ('uf1c01', uf1c01) ] #, ('uf1b01', uf1b01) ]

# 4 個 Uf 隨機排序
shuffle (uf1s)

# 看看 dj1 是否能做出正確判斷
for name, orcl in uf1s:
    print ()
    print (name + ": " + dj1 (orcl))

```

The best backend is ibmq_belem 5 qubit(s)
 Job Status: job has successfully run
 uf1c01: constant

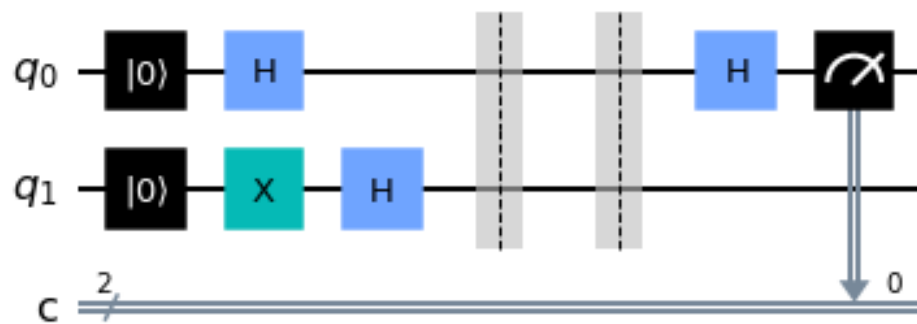
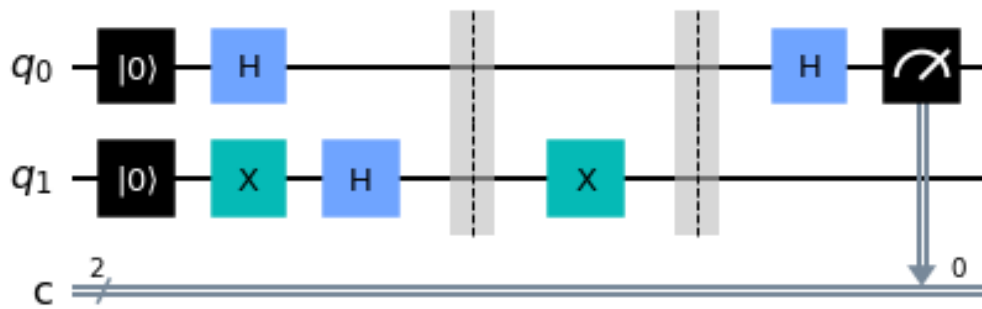
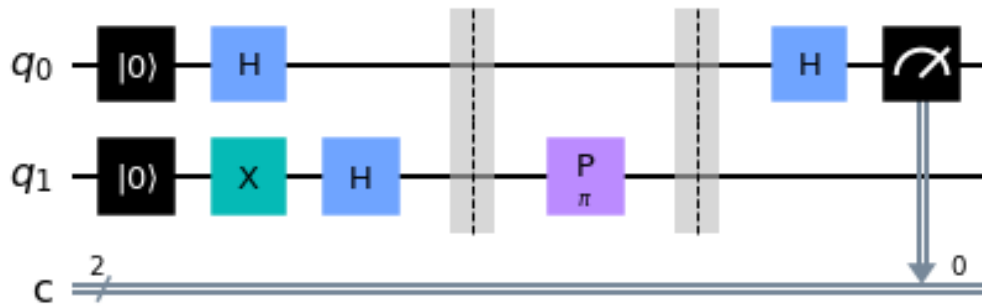
The best backend is ibmq_belem 5 qubit(s)
 Job Status: job has successfully run
 uf1c10: constant

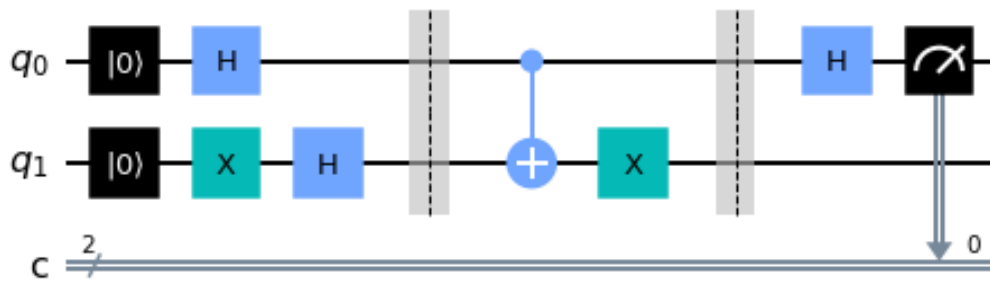
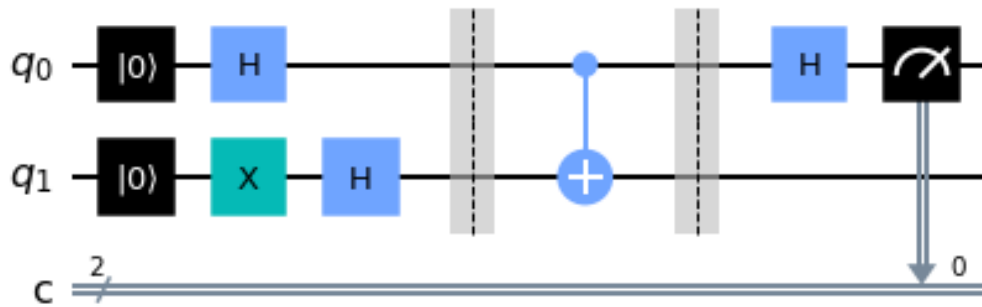
The best backend is ibmq_belem 5 qubit(s)
 Job Status: job has successfully run
 uf1c00: constant

The best backend is ibmq_belem 5 qubit(s)
 Job Status: job has successfully run
 uf1b00: balanced

The best backend is ibmq_belem 5 qubit(s)

Job Status: job has successfully run
uf1b10: balanced





0.0.1 Study CZ Oracle

```
[2]: from qiskit.quantum_info import Statevector
      #from qiskit.visualization import plot_bloch_multivector
      #from qiskit.visualization import array_to_latex

      qc = EzQC (2)
      qc.set_qubits([(0, 0), (1, 1)], bb = False)
      #plot_bloch_multivector(Statevector.from_instruction(qc), title="cz",
      #↪reverse_bits=False);
      #array_to_latex(Statevector.from_instruction(qc), prefix="\\text{\\|\\psi_0 = }")
      Statevector.from_instruction(qc).draw("latex", prefix="\\|\\psi_0\\rangle = ")
```

[2]:

$$|\psi_0\rangle = [0 \ 0 \ 1 \ 0]$$

$q_1 \otimes q_0$, swap the middle to match $q_0 \otimes q_1$

```
[3]: qc.h (0)
      qc.h (1)
      qc.barrier ()
      Statevector.from_instruction(qc).draw("latex", prefix="|\psi_1\\rangle = ")
```

[3]:

$$|\psi_1\rangle = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$

```
[4]: qc.cz (0, 1)
      qc.barrier ()
      Statevector.from_instruction(qc).draw("latex", prefix="|\psi_2\\rangle = ")
```

[4]:

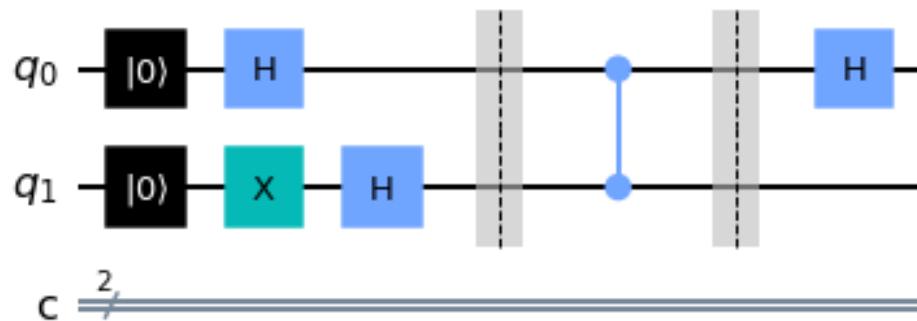
$$|\psi_2\rangle = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

```
[5]: qc.h (0)
      Statevector.from_instruction(qc).draw("latex", prefix="|\psi_3\\rangle = ")
```

[5]:

$$|\psi_3\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

```
[7]: qc.draw("mpl");
```



2.1.2 Deutsch-Josza Algorithm with 2 Qubits

1. bring all qubits to superpositions

$$|\psi_1\rangle = (H \otimes H \otimes H) |001\rangle = \frac{1}{\sqrt{2^3}}(+|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle)$$

Constant

For $f(x^{\otimes 2}) = |0\rangle$,

2. apply oracle

$$|\psi_2\rangle = |\psi_1\rangle$$

3. apply $H^{\otimes 2}$ to the data qubits

$$|\psi_3\rangle = (H \otimes H \otimes I) |\psi_2\rangle = \frac{1}{\sqrt{2^1}}(+|000\rangle - |001\rangle)$$

For $f(x^{\otimes 2}) = |1\rangle$,

2. apply oracle

$$|\psi_2\rangle = (I \otimes I \otimes X) |\psi_1\rangle = \frac{1}{\sqrt{2^3}}(-|000\rangle + |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle)$$

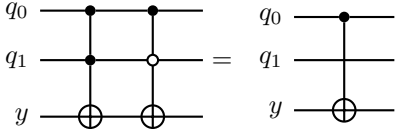
3. apply $H^{\otimes 2}$ to the data qubits

$$|\psi_3\rangle = \frac{1}{\sqrt{2^1}}(-|000\rangle + |001\rangle)$$

Balanced

For $f(|11\rangle) = f(|10\rangle) = |1\rangle$

Note: $y \oplus |1\rangle = \bar{y}$, so the oracle is



(Note this is just the oracle, it needs to be plugged into the circuit with the Hadamard gates before and after according to the Deutsch-Josza algorithm)

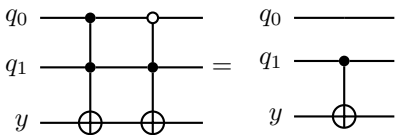
2. apply oracle

$$|\psi_2\rangle = \frac{1}{\sqrt{2^3}}(+|000\rangle - |001\rangle + |010\rangle - |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle)$$

3. apply $H^{\otimes 2}$ to the data qubits

$$|\psi_3\rangle = \frac{1}{\sqrt{2^1}}(+|100\rangle - |101\rangle)$$

For $f(|11\rangle) = f(|01\rangle) = |1\rangle$



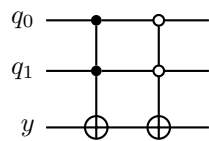
2. apply oracle

$$|\psi_2\rangle = \frac{1}{\sqrt{2^3}}(+|000\rangle - |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle)$$

3. apply $H^{\otimes 2}$ to the data qubits

$$|\psi_3\rangle = \frac{1}{\sqrt{2^1}}(+|010\rangle - |011\rangle)$$

For $f(|11\rangle) = f(|00\rangle) = |1\rangle$



2. apply oracle

$$|\psi_2\rangle = \frac{1}{\sqrt{2^3}}(-|000\rangle + |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle)$$

3. apply $H^{\otimes 2}$ to the data qubits

$$|\psi_3\rangle = \frac{1}{\sqrt{2^1}}(-|110\rangle + |111\rangle)$$

Next Page: Matrix Works

dj2

October 10, 2021

0.1 Two-Qubit Deutsch-Josza

2021 (CC BY-NC-SA 4.0) Elton Huang

```
[1]: from importlib import reload
import numpy as np
from IPython.display import display, Math
import qmtx as qm
import ezqc
reload (qm)
reload (ezqc)
```

```
[1]: <module 'ezqc' from '/Volumes/extra0/elton4k12/spqc/ezqc.py'>
```

0.2 Constant

0.2.1 For $f(x^{\otimes 2}) = |0\rangle$,

1. bring all qubits to superpositions

```
[2]: psi1 = qm.kron3 (np.dot (qm.h1, qm.ket0), np.dot (qm.h1, qm.ket0), np.dot (qm.
↪h1, qm.ket1))
qm.ket3 (psi1, "\\psi_1")
```

$$|\psi_1\rangle = \frac{1}{\sqrt{2^3}}(|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle)$$

2. apply orcale

```
[3]: psi2 = np.dot (qm.kron3 (qm.i1, qm.i1, qm.i1), psi1)
qm.ket3 (psi2, "\\psi_2")
```

$$|\psi_2\rangle = \frac{1}{\sqrt{2^3}}(|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle)$$

3. apply $H^{\otimes 2}$ to the data qubits

```
[4]: psi3 = np.dot (qm.kron3 (qm.h1, qm.h1, qm.i1), psi1)
qm.ket3 (psi3, "\\psi_3")
```

$$|\psi_3\rangle = \frac{1}{\sqrt{2^1}}(|000\rangle - |001\rangle)$$

0.2.2 For $f(x^{\otimes 2}) = |1\rangle$

2. apply oracle

```
[5]: psi2 = np.dot (qm.kron3 (qm.i1, qm.i1, qm.x1), psi1)
      qm.ket3 (psi2, "\\psi_2")
```

$$|\psi_2\rangle = \frac{1}{\sqrt{2^3}}(-|000\rangle + |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle)$$

3. apply $H^{\otimes 2}$ to the data qubits

```
[6]: psi3 = np.dot (qm.kron3 (qm.h1, qm.h1, qm.i1), psi2)
      qm.ket3 (psi3, "\\psi_3")
```

$$|\psi_3\rangle = \frac{1}{\sqrt{2^1}}(-|000\rangle + |001\rangle)$$

0.3 Balanced

0.3.1 For $f(|11\rangle) = f(|10\rangle) = |1\rangle$

note: $y \oplus |1\rangle = \bar{y}$

2. apply oracle

```
[7]: psi2 = np.dot (qm.cox, np.dot (qm.ccx, psi1))
      qm.ket3 (psi2, "\\psi_2", raw=False)
```

$$|\psi_2\rangle = \frac{1}{\sqrt{2^3}}(+|000\rangle - |001\rangle + |010\rangle - |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle)$$

```
[8]: # simplified
      psi2s = np.dot (qm.cix, psi1)
      qm.ket3 (psi2s, "\\psi_{2s}", raw=False)
```

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

3. apply $H^{\otimes 2}$ to the data qubits

```
[9]: psi3 = np.dot (qm.kron3 (qm.h1, qm.h1, qm.i1), psi2)
      qm.ket3 (psi3, "\\psi_3", raw=False)
```

$$|\psi_3\rangle = \frac{1}{\sqrt{2^1}}(+|100\rangle - |101\rangle)$$

0.3.2 For $f(|11\rangle) = f(|01\rangle) = |1\rangle$

2. apply oracle

```
[10]: psi2 = np.dot (qm.ocx, np.dot (qm.ccx, psi1))
      qm.ket3 (psi2, "\\psi_2", raw=False)
```

$$|\psi_2\rangle = \frac{1}{\sqrt{2^3}}(+|000\rangle - |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle)$$

```
[11]: qm.ket3 (np.dot (qm.icx, psi1), "\\psi_{2s}", raw=False)
      # simplified
```

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

3. apply $H^{\otimes 2}$ to the data qubits

```
[12]: psi3 = np.dot (qm.kron3 (qm.h1, qm.h1, qm.i1), psi2)
      qm.ket3 (psi3, "\\psi_3", raw=False)
```

$$|\psi_3\rangle = \frac{1}{\sqrt{2^1}}(+|010\rangle - |011\rangle)$$

0.3.3 For $f(|11\rangle) = f(|00\rangle) = |1\rangle$

2. apply oracle

```
[13]: psi2 = np.dot (qm.ooc, np.dot (qm.ccx, psi1))
      qm.ket3 (psi2, "\\psi_2", raw=False)
```

$$|\psi_2\rangle = \frac{1}{\sqrt{2^3}}(-|000\rangle + |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle)$$

3. apply $H^{\otimes 2}$ to the data qubits

```
[14]: psi3 = np.dot (qm.kron3 (qm.h1, qm.h1, qm.i1), psi2)
      qm.ket3 (psi3, "\\psi_3", raw=False)
```

$$|\psi_3\rangle = \frac{1}{\sqrt{2^1}}(-|110\rangle + |111\rangle)$$

... 3-Qubit Matrix Work (Next Page)

3qbgates

October 10, 2021

0.1 3-Qubit Gate Matrix Works

2021 (CC BY-NC-SA 4.0) Elton Huang

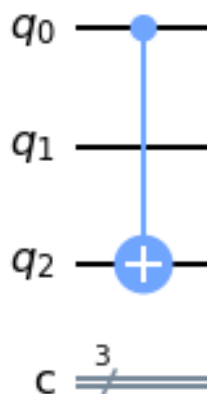
```
[1]: from importlib import reload
import numpy as np
from IPython.display import display, Math
import qmtx as qm
import ezqc
reload (qm)
reload (ezqc)
```

```
[1]: <module 'ezqc' from '/Volumes/extra0/elton4k12/spqc/ezqc.py'>
```

What are the matrices for the following 3-qubit gate
($q_i = \alpha_i|0\rangle + \beta_i|1\rangle$)

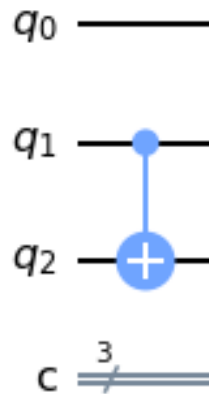
0.1.1 Gate 1

```
[2]: cix = ezqc.EzQC (3)
cix.cx (0, 2)
cix.draw (output="mpl"); # semicolon to workaround draw twice bug
```



For the circuit below,

```
[3]: icx = ezqc.EzQC (3)
      icx.cx (1, 2)
      icx.draw ("mpl");
```



the corresponding matrix follows ...

```
[4]: icxm = np.kron (qm.i1, qm.cx)
      qm.dumpgates ([icxm])
```

$$\begin{pmatrix} +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_0 \alpha_1 \alpha_2 \\ \alpha_0 \alpha_1 \beta_2 \\ \alpha_0 \beta_1 \alpha_2 \\ \alpha_0 \beta_1 \beta_2 \\ \beta_0 \alpha_1 \alpha_2 \\ \beta_0 \alpha_1 \beta_2 \\ \beta_0 \beta_1 \alpha_2 \\ \beta_0 \beta_1 \beta_2 \end{pmatrix}$$

Switching q_0 and q_1 :

```
[5]: # (this in markdown cell failed conversion to pdf)
      display (Math (r"\begin{pmatrix} 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{pmatrix} \begin{pmatrix} \alpha_0 \alpha_1 \alpha_2 \\ \alpha_0 \alpha_1 \beta_2 \\ \alpha_0 \beta_1 \alpha_2 \\ \alpha_0 \beta_1 \beta_2 \\ \beta_0 \alpha_1 \alpha_2 \\ \beta_0 \alpha_1 \beta_2 \\ \beta_0 \beta_1 \alpha_2 \\ \beta_0 \beta_1 \beta_2 \end{pmatrix}"))
```

$$\begin{pmatrix} 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{pmatrix} \begin{pmatrix} 000 \\ 001 \\ 100 \\ 101 \\ 010 \\ 011 \\ 110 \\ 111 \end{pmatrix}$$

Effectively rows 3 and 5 switch and rows 4 and 6 switch respectively.

So the matrix for qc is qc0 with rows ~~columns~~ and 3 and 5 switch and row ~~columns~~ 4 and 6 switch respectively. (striked after verified with Qiskit)

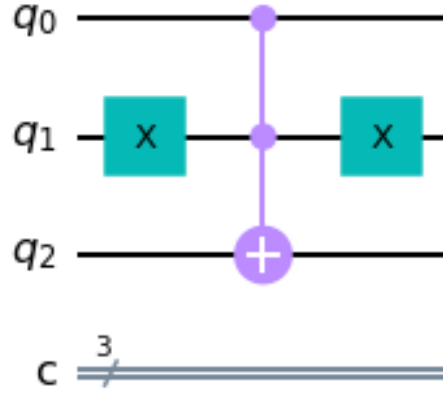
So the matrix for cix is ...

```
[6]: cixm = icxm.copy()
cixm [[2, 4], :] = cixm [[4, 2], :]
cixm [[3, 5], :] = cixm [[5, 3], :]
qm.dumpgates ([cixm])
```

$$\begin{pmatrix} +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_0 \alpha_1 \alpha_2 \\ \alpha_0 \alpha_1 \beta_2 \\ \alpha_0 \beta_1 \alpha_2 \\ \alpha_0 \beta_1 \beta_2 \\ \beta_0 \alpha_1 \alpha_2 \\ \beta_0 \alpha_1 \beta_2 \\ \beta_0 \beta_1 \alpha_2 \\ \beta_0 \beta_1 \beta_2 \end{pmatrix}$$

0.1.2 Gate 2: cox

```
[7]: cox = ezqc.EzQC (3)
cox.x (1)
cox.ccx (0, 1, 2)
cox.x (1)
cox.draw ("mpl");
```



```
[8]: ixim = qm.kron3 (qm.i1, qm.x1, qm.i1)
qm.dumpgates ([ixim, qm.ccx, ixim])
```

$$\begin{pmatrix} 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 \\ +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 \\ 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 \end{pmatrix} \begin{pmatrix} +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 \end{pmatrix} \begin{pmatrix} 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 \\ +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 \end{pmatrix} +$$

```
[9]: front2m = np.dot (ixim, qm.ccx)
qm.dumpgates ([front2m, ixim])
```

$$\begin{pmatrix} 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 \\ +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 \\ +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 \\ 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \alpha_0 \alpha_1 \alpha_2 \\ \alpha_0 \alpha_1 \beta_2 \\ \alpha_0 \beta_1 \alpha_2 \\ \alpha_0 \beta_1 \beta_2 \\ \beta_0 \alpha_1 \alpha_2 \\ \beta_0 \alpha_1 \beta_2 \\ \beta_0 \beta_1 \alpha_2 \\ \beta_0 \beta_1 \beta_2 \end{pmatrix}$$

cox

```
[10]: coxm = np.dot (front2m, ixim)
qm.dumpgates ([coxm])
```

$$\begin{pmatrix} +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 \end{pmatrix} \begin{pmatrix} \alpha_0 \alpha_1 \alpha_2 \\ \alpha_0 \alpha_1 \beta_2 \\ \alpha_0 \beta_1 \alpha_2 \\ \alpha_0 \beta_1 \beta_2 \\ \beta_0 \alpha_1 \alpha_2 \\ \beta_0 \alpha_1 \beta_2 \\ \beta_0 \beta_1 \alpha_2 \\ \beta_0 \beta_1 \beta_2 \end{pmatrix}$$

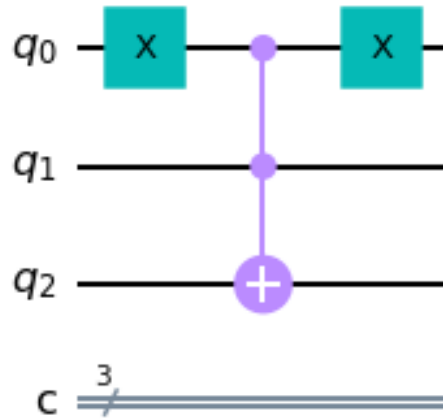
compare ccx

```
[11]: qm.dumpgates ([qm.ccx])
```

$$\begin{pmatrix} +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_0 \alpha_1 \alpha_2 \\ \alpha_0 \alpha_1 \beta_2 \\ \alpha_0 \beta_1 \alpha_2 \\ \alpha_0 \beta_1 \beta_2 \\ \beta_0 \alpha_1 \alpha_2 \\ \beta_0 \alpha_1 \beta_2 \\ \beta_0 \beta_1 \alpha_2 \\ \beta_0 \beta_1 \beta_2 \end{pmatrix}$$

0.1.3 Gate 3: ccx

```
[12]: cox = ezqc.EzQC (3)
cox.x (0)
cox.ccx (0, 1, 2)
cox.x (0)
cox.draw ("mpl");
```

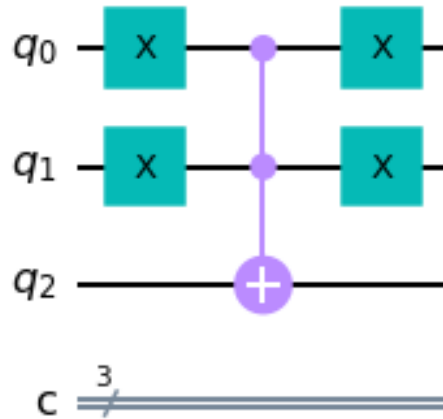


```
[13]: xiim = qm.kron3 (qm.x1, qm.i1, qm.i1)
front2m = np.dot (xiim, qm.ccx)
qm.dumpgates ([np.dot (front2m, xiim)])
```

$$\begin{pmatrix} +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 \end{pmatrix} \begin{pmatrix} \alpha_0 \alpha_1 \alpha_2 \\ \alpha_0 \alpha_1 \beta_2 \\ \alpha_0 \beta_1 \alpha_2 \\ \alpha_0 \beta_1 \beta_2 \\ \beta_0 \alpha_1 \alpha_2 \\ \beta_0 \alpha_1 \beta_2 \\ \beta_0 \beta_1 \alpha_2 \\ \beta_0 \beta_1 \beta_2 \end{pmatrix}$$

0.1.4 Gate 4: oox

```
[14]: cox = ezqc.EzQC (3)
cox.x (0)
cox.x (1)
cox.ccx (0, 1, 2)
cox.x (0)
cox.x (1)
cox.draw ("mpl");
```



```
[15]: xxim = qm.kron3 (qm.x1, qm.x1, qm.i1)
front2m = np.dot (xxim, qm.ccx)
qm.dumpgates ([np.dot (front2m, xxim)])
```

$$\begin{pmatrix} 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 \end{pmatrix} \begin{pmatrix} \alpha_0 \alpha_1 \alpha_2 \\ \alpha_0 \alpha_1 \beta_2 \\ \alpha_0 \beta_1 \alpha_2 \\ \alpha_0 \beta_1 \beta_2 \\ \beta_0 \alpha_1 \alpha_2 \\ \beta_0 \alpha_1 \beta_2 \\ \beta_0 \beta_1 \alpha_2 \\ \beta_0 \beta_1 \beta_2 \end{pmatrix}$$

... Implementation of 2-Qubit Deutsch-Josza with Qiskit (Next Page)

dj2qskt

October 10, 2021

Qiskit 實作：Deutsch-Josza
2021 (CC BY-NC-SA 4.0) Elton Huang

```
[1]: from ezqc import *

# 定義 the Deutsch's Algorithm

def dj2b (uf, shots = 1): # returns True if balanced
    qc = EzQC (3)
    qc.set_qubits([(0, 0), (1, 0), (2, 1)], bb = False)
    for i in range (3):
        qc.h (i)
    qc.barrier ()
    uf (qc)
    qc.barrier ()
    qc.h (0)
    qc.h (1)
    qc.measure (0, 0)
    qc.measure (1, 1)
    if shots == 1:
        runres = sorted (run_sim (qc, shots = shots).items(), key=lambda item:
→item[1])
    else:
        runres = sorted (run_real (qc, shots = shots).items(), key=lambda item:
→item[1])
        # 跑數次取多數決，因為會有雜訊所造成的誤差了
        # 會回傳: {'0xx': 1} (results of lower qubit indices, top of circuit
→plots, on the right)
        # q2 is of value ket-0 due to no measurement (Todo: verify, done -
→measure 1)
        print (runres, "\n")
    return (False if runres [-1][0] == '000' else True)
```

name 名稱	function 函數	prpperty 性質
uf2c0	$f(x^{\otimes 2}) = 0\rangle$	constant
uf2c1	$f(x^{\otimes 2}) = 1\rangle$	constnat
uf2b101	$f(11\rangle) = f(10\rangle) = 1\rangle$	balanced
uf2b101s	$f(11\rangle) = f(10\rangle) = 1\rangle$	balanced
uf2b011	$f(11\rangle) = f(01\rangle) = 1\rangle$	balanced
uf2b011s	$f(11\rangle) = f(01\rangle) = 1\rangle$	balanced
uf2b001	$f(11\rangle) = f(00\rangle) = 1\rangle$	balanced
uf2b100	$f(11\rangle) = f(10\rangle) = 0\rangle$	balanced
uf2b100s	$f(11\rangle) = f(10\rangle) = 0\rangle$	balanced
uf2b010	$f(11\rangle) = f(01\rangle) = 0\rangle$	balanced
uf2b010s	$f(11\rangle) = f(01\rangle) = 0\rangle$	balanced
uf2b000	$f(11\rangle) = f(00\rangle) = 0\rangle$	balanced

[2]: # 定義 1 qubit 的 4 種 U_f (oracles)

```
def uf2c0 (qc): # constant
    pass

def uf2c1 (qc): # constant
    qc.x (2)

def uf2b101 (qc): # balanced
    qc.ccx (0, 1, 2)
    qc.x (1)
    qc.ccx (0, 1, 2)
    qc.x (1)

def uf2b100 (qc): # balanced
    uf2b101 (qc)
    qc.x (2)

def uf2b011 (qc): # balanced
    qc.ccx (0, 1, 2)
    qc.x (0)
    qc.ccx (0, 1, 2)
    qc.x (0)

def uf2b010 (qc): # balanced
    uf2b011 (qc)
    qc.x (2)

def uf2b101s (qc): # balanced
    qc.cx (0, 2)

def uf2b100s (qc): # balanced
```

```

uf2b101 (qc)
qc.x (2)

def uf2b011s (qc): # balanced
    qc.cx (1, 2)

def uf2b010s (qc): # balanced
    uf2b011 (qc)
    qc.x (2)

def uf2b000 (qc): # balanced
    qc.ccx (0, 1, 2)
    qc.x (0)
    qc.x (1)
    qc.ccx (0, 1, 2)
    qc.x (0)
    qc.x (1)

def uf2b001 (qc): # balanced
    uf2b000 (qc)
    qc.x (2)

```

```

[3]: from random import shuffle

# 將 4 個 Uf 存到一個 list 中，並註記名稱，以及預期結果 (True 為 balanced)
uf2s = [ ('uf2c0', uf2c0, False), ('uf2c1', uf2c1, False),
          ('uf2b101', uf2b101, True), ('uf2b100', uf2b100, True),
          ('uf2b011', uf2b011, True), ('uf2b010', uf2b010, True),
          ('uf2b101s', uf2b101s, True), ('uf2b100s', uf2b100s, True),
          ('uf2b011s', uf2b011s, True), ('uf2b010s', uf2b010s, True),
          ('uf2b000', uf2b000, True), ('uf2b001', uf2b001, True) ]

# 先用模擬驗證
for name, orcl, expb in uf2s:
    print (name + ": " + ("Correct" if dj2b (orcl) == expb else "INCORRECT"))

```

```

uf2c0: Correct
uf2c1: Correct
uf2b101: Correct
uf2b100: Correct
uf2b011: Correct
uf2b010: Correct
uf2b101s: Correct
uf2b100s: Correct
uf2b011s: Correct
uf2b010s: Correct
uf2b000: Correct
uf2b001: Correct

```

```
[4]: import sympy as sp

# 8 個 Uf 隨機排序
shuffle (uf2s)

# 看看今天跑到幾個 shots 才能全對
sh = 3
while True:
    print ("shots: " + str (sh) + "\n")
    incorrect = 0
    for name, orcl, expb in uf2s:
        print (name)
        if dj2b (orcl, shots = sh) != expb:
            incorrect += 1
            break
    # print (sh, "shots: 錯", incorrect, "個")
    if incorrect == 0:
        print (sh, "shots 完成")
        break
    else:
        sh = sp.nextprime (sh)
```

shots: 3

uf2b100s

The best backend is ibmq_belem 5 qubit(s)
 Job Status: job has successfully run
 [('000', 1), ('001', 2)]

uf2b000

The best backend is ibmq_belem 5 qubit(s)
 Job Status: job has successfully run
 [('011', 3)]

uf2b011s

The best backend is ibmq_belem 5 qubit(s)
 Job Status: job has successfully run
 [('010', 3)]

uf2b011

The best backend is ibmq_belem 5 qubit(s)
 Job Status: job has successfully run
 [('001', 1), ('010', 1), ('011', 1)]

uf2c1

The best backend is ibmq_belem 5 qubit(s)
 Job Status: job has successfully run
 [('000', 3)]

uf2b010s
The best backend is ibmq_belem 5 qubit(s)
Job Status: job has successfully run
[('000', 1), ('010', 2)]

uf2b001
The best backend is ibmq_lima 5 qubit(s)
Job Status: job has successfully run
[('011', 3)]

uf2c0
The best backend is ibmq_belem 5 qubit(s)
Job Status: job has successfully run
[('000', 3)]

uf2b100
The best backend is ibmq_belem 5 qubit(s)
Job Status: job has successfully run
[('010', 1), ('000', 2)]

shots: 5

uf2b100s
The best backend is ibmq_belem 5 qubit(s)
Job Status: job has successfully run
[('000', 1), ('001', 4)]

uf2b000
The best backend is ibmq_belem 5 qubit(s)
Job Status: job has successfully run
[('000', 1), ('001', 1), ('011', 3)]

uf2b011s
The best backend is ibmq_belem 5 qubit(s)
Job Status: job has successfully run
[('010', 5)]

uf2b011
The best backend is ibmq_belem 5 qubit(s)
Job Status: job has successfully run
[('011', 1), ('001', 2), ('010', 2)]

uf2c1
The best backend is ibmq_belem 5 qubit(s)
Job Status: job has successfully run
[('000', 5)]

uf2b010s

The best backend is ibmq_belem 5 qubit(s)

Job Status: job has successfully run

[('001', 1), ('010', 2), ('011', 2)]

uf2b001

The best backend is ibmq_belem 5 qubit(s)

Job Status: job has successfully run

[('000', 1), ('010', 2), ('011', 2)]

uf2c0

The best backend is ibmq_belem 5 qubit(s)

Job Status: job has successfully run

[('000', 5)]

uf2b100

The best backend is ibmq_belem 5 qubit(s)

Job Status: job has successfully run

[('011', 2), ('001', 3)]

uf2b101

The best backend is ibmq_belem 5 qubit(s)

Job Status: job has successfully run

[('000', 1), ('010', 1), ('011', 1), ('001', 2)]

uf2b010

The best backend is ibmq_belem 5 qubit(s)

Job Status: job has successfully run

[('010', 2), ('011', 3)]

uf2b101s

The best backend is ibmq_belem 5 qubit(s)

Job Status: job has successfully run

[('001', 5)]

5 shots 完成

2.2 More Oracle-Based

2.2.1 Bernstein-Vazirani

2.2.2 Simon

2.3 Grover's

- Why is it phase?
- How does it work?

Quantum Computing for Everyone Page 179:

- flips about the mean
- CZ, the matrix reversible operations: controlling and controlled no-differently entangled

2.4 Exercise: Superposition

$$|1\rangle + |2\rangle + |3\rangle + |7\rangle \longrightarrow \boxed{f(x) = 2x} \longrightarrow |2\rangle + |4\rangle + |6\rangle + |14\rangle$$

$$\frac{1}{2}(|0001\rangle + |0010\rangle + |0011\rangle + |0111\rangle) \longrightarrow \boxed{f(x) = 2x} \longrightarrow \frac{1}{2}(|0010\rangle + |0100\rangle + |0110\rangle + |1110\rangle)$$

Lee et. al, *Quantum Shift Register*, 2001

2.5 Quantum Circuit Synthesis

Shende et. al, *Synthesis of Quantum Logic Circuits*

Page 2 bottom

Page 5 top §2.1 end: $2^3 = 8 \gg 2^1 + 2^2 = 5$

“Much interest in quantum computing is driven by this exponential scaling of the state space, and the loss of independence between different subsystems is called *quantum entanglement*.”

Matteo, *Parallelizing quantum circuit synthesis*, 2015

Chapter 3

Applications¹

3.1 Optimization

3.1.1 Quadratic Problems with Qiskit

3.1.2 Knapsack with Ocean

3.2 Monte Carlo Method

Monte Carlo Method

¹... to appreciate the values

3.3 Chemical Simulation

“every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means’. Classical physics and the universal Turing machine, because the former is continuous and the latter discrete, do not obey the principle, at least in the strong form above.” - D. Deutsch 1985

Exploring greener approaches to nitrogen fixation: 103 electrons in 71 orbitals

Protein crystallization

- sequence \rightarrow predict shapes
- shape analysis
- hybrid: e.g. 高能物理的方式預測 interactions

3.3.1 Molecular Dynamics with Qiskit

3.3.2 Published Works

On the construction of model Hamiltonians for adiabatic quantum computation and its application to finding low energy conformations of lattice protein models, 2008

Finding low-energy conformations of lattice protein models by quantum annealing, 2012

Quantum annealing versus classical machine learning applied to a simplified computational biology problem, 2018

Quantum Molecular Unfolding, Mente AI, 2021

- shape, reactivity: shape \rightarrow reactivity \rightarrow energy cost
- approach assumption: pocket rigid, ligand flexible
- fixed length chemical bonds with a subset *rotatable* (torsionals) which split the molecule in two nonempty disjointed fragments, when virtually removed
- 3 phases in Molecular Docking: (1) *Ligand expansion*, (2) Initial Placement and (3) Shape Refinement inside the pocket

Molecular Unfolding find the unfolded shape of the ligand (torsion configuration) that maximizes the total sum of internal distances between pairs of atoms in the ligand (molecular volume).

Designing Peptides on a Quantum Computer, 2019

Förster resonance energy transfer: Role of diffusion of fluorophore orientation and separation in observed shifts of FRET efficiency, 2017

Efficient quantum simulation of photosynthetic light harvesting, 2018

3.3.3 OpenFermion

3.4 Machine Learning

Quantum's advantage solves black box bit riddle

Demonstration of quantum advantage in machine learning

3.4.1 TensorFlow Quantum

3.4.2 Qiskit ML

Chapter 4

Explore and Exercise

4.1 IBM Quantum *Errors*

Noise is one of the utmost problems with Quantum Computer hardware technology presently.

This exercise demonstrates how to check the map of error rates for each qubits in an IBM Qunatum computer with Qiskit and a brief example to verify it.

Further indepth exercise may involve Qiskit-Pulse, the pulse-level programming kit.

(Next page)

errors

October 7, 2021

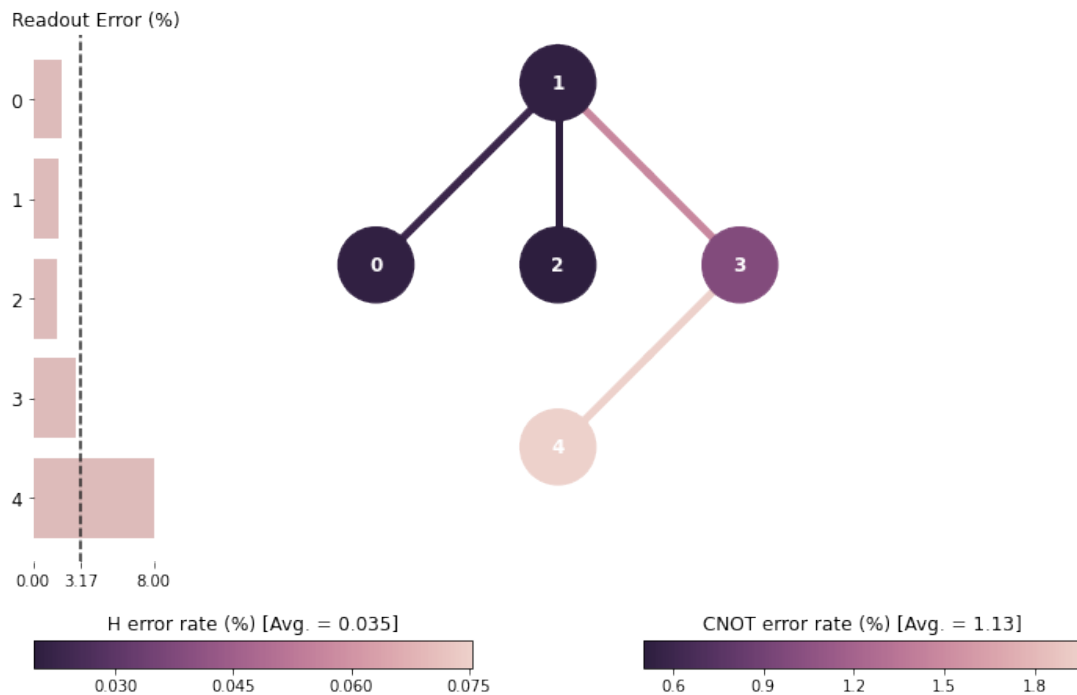
Explore and Exercise: *Errors*

2021 (CC BY-NC-SA 4.0) Elton Huang

```
[1]: from qiskit import *  
from qiskit.visualization import plot_error_map  
IBMQ.load_account() # ezqc.SetCloud()  
backend = IBMQ.get_provider('ibm-q').get_backend('ibmq_lima')  
plot_error_map(backend)
```

[1]:

ibmq_lima Error Map



`qiskit.compiler.transpile`: initial_layout

`qiskit.result.Result.get_counts`: measured states of the vector of qubits

```
[2]: from qiskit.tools.monitor import job_monitor

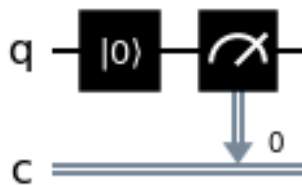
def runiton (qc, res, phqb):
    global backend
    qc_trans = transpile(qc, backend, initial_layout=[phqb],
    ↪optimization_level=3)
    job = backend.run(qc_trans, shots=8192)
    # print(job.job_id())
    job_monitor(job)
    output = job.result().get_counts()
    print('Run with qubit #' + str (phqb))
    print('Probability of correct answer : {:.2f}'.format(output[res]/8192))
```

```
[3]: qc0 = QuantumCircuit(1, 1)
      qc0.reset(0)
      qc0.measure(0, 0)

      qc1 = QuantumCircuit(1, 1)
      qc1.reset(0)
      qc1.x (0)
      qc1.measure(0, 0)
```

```
[3]: <qiskit.circuit.instructionset.InstructionSet at 0x7f1760070d00>
```

```
[4]: for qc, res in [ (qc0, '0'), (qc1, '1') ]:
      display (qc.draw())
      for i in range (backend.configuration().n_qubits):
          runiton (qc, res, i)
```



```
Job Status: job has successfully run
Run with qubit #0
Probability of correct answer : 0.99
Job Status: job has successfully run
Run with qubit #1
Probability of correct answer : 1.00
```

Job Status: job has successfully run
 Run with qubit #2
 Probability of correct answer : 1.00
 Job Status: job has successfully run
 Run with qubit #3
 Probability of correct answer : 0.99
 Job Status: job has successfully run
 Run with qubit #4
 Probability of correct answer : 0.99



Job Status: job has successfully run
 Run with qubit #0
 Probability of correct answer : 0.96
 Job Status: job has successfully run
 Run with qubit #1
 Probability of correct answer : 0.97
 Job Status: job has successfully run
 Run with qubit #2
 Probability of correct answer : 0.97
 Job Status: job has successfully run
 Run with qubit #3
 Probability of correct answer : 0.95
 Job Status: job has successfully run
 Run with qubit #4
 Probability of correct answer : 0.82

4.2 Virtual Quantum Optics Laboratory

The Virtual Quantum Optics Laboratory

Appendices

Appendix A

ezqc.py

```
1  # EzQC Library
2  # 2021 (CC BY-NC-SA 4.0) Elton Huang
3
4  import numpy as np
5  from qiskit import * # QuantumCircuit, transpile, IBMQ
6  from qiskit.providers.aer import QasmSimulator
7  # from qiskit.visualization import plot_histogram
8  import matplotlib.pyplot as plt
9
10 from qiskit.providers.ibmq import least_busy
11 from qiskit.tools.monitor import job_monitor
12
13 bCloudYet2Set = True
14
15 def SetCloud ():
16     global provider, bCloudYet2Set
17     while True:
18         try:
19             # print ("loading IBMQ account ...")
20             provider = IBMQ.load_account()
21             break
22         except:
23             IBMQ.save_account (input ("IBMQ Token:"))
24     bCloudYet2Set = False
25
26 def run_sim (circuit, shots = 1024):
27     simulator = QasmSimulator()
28     compiled_circuit = transpile (circuit, simulator)
29     counts = simulator.run (compiled_circuit, shots = shots).result().get_counts (compiled_circuit)
30     # print("\nResults:",counts)
31     return (counts)
32
33 def run_real (circuit, shots = 1024): # real hw?
34     global bCloudYet2Set
35     if 'google.colab' in sys.modules:
36         print ("You are on Google Colab. Switching to simulation ...")
37         return (run_sim (circuit))
38     else:
39         if bCloudYet2Set:
40             SetCloud ()
41         devices = provider.backends (filters = lambda x: x.configuration().n_qubits > circuit.num_qubits
42                                     and not x.configuration().simulator
43                                     and x.name() not in ['ibmq_bogota'], simulator = False)
44
45         # bogota: borken
46         if len (devices) == 0: # not necessary, bm: sim_stab 200s, colab/min 2mins
47             devices = provider.backends (filters = lambda x: x.configuration().n_qubits > circuit.num_qubits
```

```

47         and x.name() not in ['ibmq_bogota'], simulator = False)
48     # print (devices)
49     backend = least_busy (devices)
50     print ("The best backend is", backend, backend.configuration().n_qubits, "qubit(s)")
51     job = execute (circuit, backend = backend, shots = shots)
52     job_monitor (job, interval = 5)
53     result = job.result()
54     counts = result.get_counts(circuit)
55     # print("\nResults:", counts)
56     return (counts)
57
58 def plot (counts):
59     fig = plt.figure(figsize = (10, 5)) # default 6.4x4.8 inches
60     ax = fig.add_axes([0,0,1,1])
61     x = sorted (counts.keys(), key=lambda x: int(x, 2))
62     y = list (counts [i] for i in x)
63     ax.bar (x, y, width = 0.5, color = 'cornflowerblue')
64     ax.set_ylabel('Frequency')
65     ax.set_xlabel('Measurement Output')
66     for i, v in enumerate (y):
67         ax.text(i - .25, v + 3, str (v) + " / " + str (round (100 * v / sum (y))) + "%", color='blue', fontweight='bold')
68     plt.show()
69
70 class EzQC (QuantumCircuit):
71
72     def __init__ (self, qbn):
73         super().__init__ (qbn, qbn)
74
75     # fb = True to add front barrier
76     def __fb (self, **kwargs):
77         if 'fb' in kwargs.keys() and kwargs.get ('fb') == True:
78             self.barrier ()
79
80     # bb = False to omit back barrier
81     def __bb (self, **kwargs):
82         if 'bb' not in kwargs.keys() or kwargs.get ('bb') != False:
83             self.barrier ()
84
85     def set_qubits (self, qubit_ketv_pair_list, **kwargs):
86         self.__fb (**kwargs)
87         for qb, kv in qubit_ketv_pair_list:
88             self.reset (qb)
89             if kv == 1:
90                 self.x (qb)
91         self.__bb (**kwargs)
92
93     # reset = False for no reset to ket-0
94     def ghz (self, list3qbs, **kwargs):
95         self.__fb (**kwargs)
96         if 'reset' not in kwargs.keys() or kwargs.get ('reset') != False:
97             for i in list3qbs:
98                 self.reset (i)
99         self.h (list3qbs [0])
100         self.cx (list3qbs [0], list3qbs [1])
101         self.cx (list3qbs [0], list3qbs [2])
102         self.__bb (**kwargs)
103
104     def cox (self, q0, q1, q2, **kwargs):
105         self.__fb (**kwargs)
106         self.x (q1)
107         self.ccx (q0, q1, q2)
108         self.x (q1)
109         self.__bb (**kwargs)
110

```

```
111     def measure_all (self, **kwargs):
112         self.__fb (**kwargs)
113         self.measure (list (range (self.num_qubits)), list (range (self.num_qubits)))
```

Appendix B

qmtx.py

```
1  # qmtx Library
2  # 2021 (CC BY-NC-SA 4.0) Elton Huang
3
4  import numpy as np
5  # from itertools import product
6  from IPython.display import display, Math
7  import math
8
9  b3string = ['|000\\rangle', '|001\\rangle', '|010\\rangle', '|011\\rangle',
10             '|100\\rangle', '|101\\rangle', '|110\\rangle', '|111\\rangle']
11
12  b3kets = ['\\ket{000}', '\\ket{001}', '\\ket{010}', '\\ket{011}',
13            '\\ket{100}', '\\ket{101}', '\\ket{110}', '\\ket{111}']
14
15  opr3 = "\\begin{pmatrix} \\alpha_0\\alpha_1\\alpha_2 & \\alpha_0\\alpha_1\\beta_2 & \\alpha_0\\beta_1 & \\alpha_2 & \\beta_0 & \\alpha_1\\alpha_2 & \\beta_0 & \\alpha_1\\beta_2 & \\beta_0 & \\beta_1 & \\beta_2 \\\\
16         & \\alpha_0\\beta_1 & \\alpha_2 & \\alpha_0\\beta_1 & \\beta_2 & \\beta_0 & \\alpha_1\\alpha_2 & \\beta_0 & \\alpha_1\\beta_2 & \\beta_0 & \\beta_1 & \\beta_2 \\\\
17         & \\beta_0 & \\alpha_1\\alpha_2 & \\beta_0 & \\alpha_1\\beta_2 & \\beta_0 & \\beta_1 & \\beta_2 & \\beta_0 & \\beta_1 & \\beta_2 \\\\
18         & \\beta_0 & \\beta_1 & \\alpha_2 & \\beta_0 & \\beta_1 & \\beta_2 & \\end{pmatrix}"
19
20  ket0 = np.array ( [[1],
21                    [0]] )
22
23  ket1 = np.array ( [[0],
24                    [1]] )
25
26  h1 = 1/np.sqrt(2)*np.array([[1, 1],
27                             [1,-1]])
28
29  i1 = np.array ([[1, 0],
30                 [0, 1]])
31
32  x1 = np.array ([[0, 1],
33                 [1, 0]])
34
35  cx = np.array ([[1, 0, 0, 0],
36                 [0, 1, 0, 0],
37                 [0, 0, 0, 1],
38                 [0, 0, 1, 0]])
39
40  ccx = np.array ([[1, 0, 0, 0, 0, 0, 0, 0],
41                  [0, 1, 0, 0, 0, 0, 0, 0],
42                  [0, 0, 1, 0, 0, 0, 0, 0],
43                  [0, 0, 0, 1, 0, 0, 0, 0],
44                  [0, 0, 0, 0, 1, 0, 0, 0],
45                  [0, 0, 0, 0, 0, 1, 0, 0],
46                  [0, 0, 0, 0, 0, 0, 1, 0],
47                  [0, 0, 0, 0, 0, 0, 0, 1]])
```

```

47         [0, 0, 0, 0, 0, 0, 1, 0]])
48
49     cox = np.array ([[1, 0, 0, 0, 0, 0, 0, 0],
50                     [0, 1, 0, 0, 0, 0, 0, 0],
51                     [0, 0, 1, 0, 0, 0, 0, 0],
52                     [0, 0, 0, 1, 0, 0, 0, 0],
53                     [0, 0, 0, 0, 1, 0, 0, 0],
54                     [0, 0, 0, 0, 1, 0, 0, 0],
55                     [0, 0, 0, 0, 0, 1, 0, 0],
56                     [0, 0, 0, 0, 0, 0, 1, 0]])
57
58     ocx = np.array ([[1, 0, 0, 0, 0, 0, 0, 0],
59                     [0, 1, 0, 0, 0, 0, 0, 0],
60                     [0, 0, 0, 1, 0, 0, 0, 0],
61                     [0, 0, 1, 0, 0, 0, 0, 0],
62                     [0, 0, 0, 0, 1, 0, 0, 0],
63                     [0, 0, 0, 0, 0, 1, 0, 0],
64                     [0, 0, 0, 0, 0, 0, 1, 0],
65                     [0, 0, 0, 0, 0, 0, 0, 1]])
66
67     oox = np.array ([[0, 1, 0, 0, 0, 0, 0, 0],
68                     [1, 0, 0, 0, 0, 0, 0, 0],
69                     [0, 0, 1, 0, 0, 0, 0, 0],
70                     [0, 0, 0, 1, 0, 0, 0, 0],
71                     [0, 0, 0, 0, 1, 0, 0, 0],
72                     [0, 0, 0, 0, 0, 1, 0, 0],
73                     [0, 0, 0, 0, 0, 0, 1, 0],
74                     [0, 0, 0, 0, 0, 0, 0, 1]])
75
76     icx = np.array ([[1, 0, 0, 0, 0, 0, 0, 0],
77                     [0, 1, 0, 0, 0, 0, 0, 0],
78                     [0, 0, 0, 1, 0, 0, 0, 0],
79                     [0, 0, 1, 0, 0, 0, 0, 0],
80                     [0, 0, 0, 0, 1, 0, 0, 0],
81                     [0, 0, 0, 0, 0, 1, 0, 0],
82                     [0, 0, 0, 0, 0, 0, 1, 0],
83                     [0, 0, 0, 0, 0, 0, 1, 0]])
84
85     cix = np.array ([[1, 0, 0, 0, 0, 0, 0, 0],
86                     [0, 1, 0, 0, 0, 0, 0, 0],
87                     [0, 0, 0, 0, 1, 0, 0, 0],
88                     [0, 0, 0, 0, 0, 1, 0, 0],
89                     [0, 0, 0, 1, 0, 0, 0, 0],
90                     [0, 0, 1, 0, 0, 0, 0, 0],
91                     [0, 0, 0, 0, 0, 0, 1, 0],
92                     [0, 0, 0, 0, 0, 0, 1, 0]])
93
94     def kron3 (q0, q1, q2): # 3 qubits tensor product
95         return np.kron (np.kron (q0, q1), q2)
96
97     def ket3 (psi, var, raw=False): # Todo: ket-n
98         lnn = 0
99         out = ""
100         for r in range (psi.shape[0]):
101             if abs (psi [r][0]) > 1e-10: # Todo: complex numbers
102                 lnn += 1
103                 out += ('-' if psi [r][0] < 0 else '+') + (b3kets [r] if raw else b3string [r]) # Todo: non-uniform coeffecients
104         if raw:
105             print (('ket{' + var.replace('\\\\', '\\') + '} = ') +
106                   (('' if lnn == 1 else
107                    ('\\frac{1}{\\sqrt{2}}' + str (int (math.log (lnn, 2))) + ')(') + out +
108                    ('' if lnn == 1 else ')')).replace('\\\\', '\\'))
109         else:
110             display (Math(("|" + var + "\\rangle = ") +

```

```

111         (" if lnn == 1 else
112         ("\\frac{1}{\\sqrt{2}}" + str (int (math.log (lnn, 2))) + "{(}") + out +
113         (" if lnn == 1 else ")"))
114
115 def dump (mtx):
116     lnn = 0
117     out = "\\begin{pmatrix} "
118     for i in range (mtx.shape[0]):
119         for j in range (mtx.shape[1]):
120             if abs (mtx [i][j]) > 1e-10: # Todo: complex numbers
121                 lnn += 1
122                 out += '-1 ' if mtx [i][j] < 0 else '+1 ' # Todo: non-uniform coeffecients
123             else:
124                 out += ' 0 '
125             if j + 1 < mtx.shape[1]:
126                 out += '& '
127         out += "\\\\"
128     out += "\\end{pmatrix}"
129     display(Math((" if lnn == 1 else
130     ("\\frac{1}{\\sqrt{2}}" + str (int (math.log (lnn, 2))) + "{(}") + out))
131
132 def dumpgates (mtx):
133     sq2n = 0
134     out = ""
135     for mtx in mtxs:
136         lnn = 0
137         out += "\\begin{pmatrix} "
138         for i in range (mtx.shape[0]):
139             for j in range (mtx.shape[1]):
140                 if abs (mtx [i][j]) > 1e-10: # Todo: complex numbers
141                     lnn += 1
142                     out += '-1 ' if mtx [i][j] < 0 else '+1 ' # Todo: non-uniform coeffecients
143                 else:
144                     out += ' 0 '
145                 if j + 1 < mtx.shape[1]:
146                     out += '& '
147             out += "\\\\"
148         out += "\\end{pmatrix}"
149         sq2n += lnn // mtx.shape[0] - 1
150     display(Math((" if sq2n == 0 else
151     ("\\frac{1}{\\sqrt{2}}" + str (sq2n) + "{(}") + out + opr3))
152
153 # https://jarrodmcclean.com/basic-quantum-circuit-simulation-in-python/

```