

Communication Systems Laboratory

Lab 1: Simulating Quantum Circuits with QISKit

(Report Due: 21:00, October 6, 2021)

1 Overview

The objective of this laboratory/experiment course is to prepare you for research in **information processing systems**, which include signal processing, classical communication, quantum computing and quantum communication. The preliminary courses are Linear Algebra, Signals and Systems, Probability and Statistics, Digital Signal Processing, Principle of Communications, Quantum Information and Computation (which I regularly teach in every Spring). Due to the time constraint, we will only cover parts of quantum computing and classical/quantum communication in the course.

The focus of the present course is different from that of the above-mentioned courses. Therefore, those courses are not necessarily required. In this course, you will get your hands dirty (i.e. coding) by simulating various systems using Matlab, LabVIEW, USRP, and the QISKit . By doing so, you can experiment with your simulation program to obtain a better understanding of the working principles behind those systems.

QISKit is a software development kit (SDK) for quantum programming in the cloud and is part of the IBM Quantum Experience cloud platform. The exercises in this lab will guide you to build QISKit scripts with good practice. We will learn how to prepare a quantum state, to manipulate quantum gates, to fetch the measured statistics, and to implement some basic quantum systems. We will experiment with the Swap test, and the Quantum Random Number Generator.

2 Experiments

QISKit packs a set of helpful simulators to execute your quantum programs locally or remotely, but it also allows you to run in the real IBM's cloud quantum processor. The problems given below will guide you step by step how to use the IBM Quantum Experience cloud platform to learn the basics of quantum circuits.

1. (20 points) Manipulating a single qubit state:

Your first aim in Lab 1 is to watch the course video "*Installing QISKit*", and properly installed it. In this problem, we will go through each step of a single qubit quantum program in detail.

The basic pseudo code can be summarized as the following:

1. Import necessary QISKit libraries.
2. Create a quantum program.
3. Create one or more qubits, and classical registers, in which we will store the outcomes of measuring the qubits.

4. Initialize a circuit which groups the qubits in a logical execution unit.
5. Apply quantum gates on the qubits to achieve a desired result.
6. Measure the qubits into the classical register to collect a final result.
7. Compile the program.
8. Run in the simulator or real quantum device.
9. Fetch the results.

Now, please follow the instructions given below to understand the purpose of each step.

- (a) Firstly, we import some libraries:

```
import qiskit
import numpy as np
from qiskit import QuantumCircuit, execute, Aer, *
from qiskit.visualization import plot_histogram, plot_bloch_vector
from math import sqrt, pi
```

Note that this step depends on what you really need in executing the program. To avoid missing some necessary libraries, always do `from qiskit import *` as a rule of thumb.

- (b) Secondly, we create two qubits and one classical bits:

```
q = QuantumRegister(2)
c = ClassicalRegister(2)
```

Here, 'q' and 'c' are the registers for qubits and classical bits, respectively. You can imagine them as the corresponding "quantum wires" and the "classical wires".

- (c) Thirdly, using the registers declared above to create a quantum circuit (i.e. a bundle of registers you can run on them):

```
circuit = QuantumCircuit(q,c)
```

Now, each initial qubit of the circuit is in the state $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. You can verify this by using the following commands to get the vector form of the state:

```
simulator = Aer.get_backend('statevector_simulator')
job = execute(circuit, simulator)
result = job.result()
statevector = result.get_statevector()
```

```
print(statevector)
plot_bloch_multivector(statevector)
```

Here, we have used the 'state_simulator' in QISKit.Aer to compute the mathematical description of the state. (Note that Python uses 'j' to denote the imaginary unit; throughout the course we will use the convention 'i'.) There are other simulators in QISKit, which can be found in <https://qiskit.org/documentation/tutorials/simulators/index.html>.

The command `plot_bloch_multivector{statevector}` plots/visualizes the Bloch-sphere representation of the state $|q\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$. You can see that both the two qubits are in the initial state $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

- (d) Now we measure all the states with respect to the computational basis $\{|0\rangle, |1\rangle\}$ ($-\square=\square$), and use 'qasm_simulator' in QISKit.Aer as the simulator:

```
circuit.measure(q,c)
circuit.draw()
simulator = Aer.get_backend('qasm_simulator')
job = execute(circuit, simulator, shots = 1024)
result = job.result()
counts = result.get_counts()
print(counts)
```

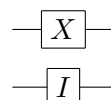
What did you get? Here, you should get each measurement outcome (stored in the classical registers) with certain probability that can be calculated by the *Born rule*.

Now I believe that you have equipped with what are needed. Follow the instructions given below and answer questions in your report.


- (e) (4 points) Let us apply some single qubit gates on the initial qubit $|0\rangle$. Please apply the 'X' gate (i.e the *NOT gate*) $-\square-$ on the first qubit by running `circuit.x(q[0])`, and draw the circuit by running `circuit.draw()`. Here, the '0' means the first register of the quantum register 'q', i.e. the first qubit.

Now, print it and plot the it on the Bloch sphere. You will see that the first qubit is now in the state $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ since it has been flipped by the 'X' gate.



Note that the second register is remained in the state $|0\rangle$ due to the following reasons: (i) the initial two qubits is a product state $|0\rangle \otimes |0\rangle$; that is, the first qubit is *independent* of the second qubit; and (ii) the quantum operation is equivalent to 'X \otimes I':



Herein the first gate is *independent* of the second gate (i.e. the identity gate). You can verify this by running `circuit.i(q[1])`.

Measure it —  — and run `plot_histogram(counts)`. What did you get, again?

There are many fundamental gates such as the 'Y' gate and the 'Z' out there. You may want to try different gates to see what's going on.

- (f) (4 points) Now, try applying a Hadamard gate —  — on the first qubit by running `circuit.h(q[0])`. Measure it —  —, and run `plot_histogram(counts)` to see your measurement outcomes. What did you get then? After applying the Hadamard gate (either on states $|0\rangle$ or $|1\rangle$) you are supposed to get a superposition of $|0\rangle$ and $|1\rangle$ with equal weights.

Remark. If you apply a Hadamard gate after another Hadamard gate, you will obtain the original state (before the first Hadamard gate). That is because the Hadamard gate is self-inverse. You can verify this either by simulation or doing the algebra by the matrix representation of the gate.

- (g) (4 points) Run `circuit.rx(math.pi/2, q[0])`. This applies the ' R_X ' gate:

$$R_X(\theta) := \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

with a certain angle, say $\theta = \pi/2$ in this example, that rotates the state along the ' x '-axis on the Bloch sphere. You can verify this by plotting it on the Bloch sphere. Similarly, you can play with the ' R_Y ' and ' R_Z ' gates.

Now since any single qubit can be represented on the Bloch sphere, you are now able to create *an arbitrary 1-qubit (pure) state* by a bunch of 1-qubit gates.

Note. There is a so-called U_3 gate in QISKit, (`circuit.u3()`):

$$U_3(\theta, \phi, \lambda) := \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & e^{i(\lambda+\phi)}\cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

parameterized by the three variables (θ, ϕ, λ) . It also allows you to reach any point on the Bloch sphere.

- (h) (8 points) Prepare a qubit state that will give a $2/3$ probability of obtaining $|-\rangle := \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ via the measurement with respect to the basis $\{|+\rangle, |-\rangle\}$. Describe the circuit you use for preparing the state in your report. Justify your method, e.g. by observing the measurement outcomes. Is your proposed state unique?

Hint. The command `circuit.measure()` denotes the measurement with respect to the computational basis. However, you may use the Hadamard gate to get the result of this problem.

Note. You can actually use `circuit.initialize` to create an arbitrary state by its mathematical description. However, I believe you can solve problem 1h without using it. Yet,

`circuit.initialize` is still a very convenient tool for numerically simulating quantum states. Just to bear in mind that *it is neither physical nor practical*. Namely, you can run it only via simulators such as `qasm_simulator` but not on a real device.

Remark. In Theorem 4.1 of Nielsen & Chuang's book, any 1-qubit gate by using the ' R_Y ' and ' R_Z ' gates merely.

(Bonus 5 points) Justify and explain why the U_3 gate can equivalently act as *any* 1-qubit gate.

Remark. The interested students can refer to https://qiskit.org/documentation/tutorials/circuits/3_summary_of_quantum_operations.html for finding the library of quantum operations available in the QISKit package.

2. (20 points) **Manipulating Multi-Qubit Gates:** In the previous problem, we have known how to apply a 1-qubit gate to prepare an arbitrary 1-qubit (pure) state. However, if we can only prepare a single qubit gate for each register of the circuit, then the operation at each register and at each state it is at most a (2×2) matrix multiplication. Moreover, the state is in the product form throughout the whole computation. It is not hard to use a classical computer to *efficiently* simulate such a quantum circuit and quantum operation.

In this problem, you will learn how to use multi-qubit gates to prepare *entangled* quantum state, which makes your quantum computation much more profound.

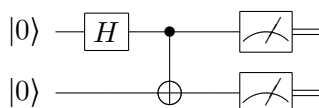
- (a) (4 Points) Apply the Hadamard gates $H \otimes H$ on the initial states $|0\rangle \otimes |0\rangle$. Measure it and what did you get?

Now you may know how to prepare a superposition (with equal weight) of all the n -bit string (which contain *exponentially many* terms) in just one n -qubit quantum state by using only *linearly many* elementary quantum gates.

- (b) (4 Points) The ' CX ' gate (also called the *control-not gate*)



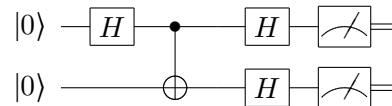
is a basic two-qubit gate that can be used to prepare an entangled state. Try to apply a Hadamard gate on the first qubit $|0\rangle$, and then to apply a control- X gate with the first register as the control qubit and the second register as the target qubit:



Now, this state $\frac{|00\rangle + |11\rangle}{2}$ can not be expressed as tensor product of any two qubits. Hence, it is called an *entangled state*. In this case, the state $|\Phi^+\rangle := \frac{|00\rangle + |11\rangle}{2}$ is called a *maximally entangled state*.

Check your measurement outcomes. If the first classical register (say, possessed by Alice) is '1', what is the outcome of the second classical register (say, possessed by Bob)? What is the probability of getting outcome '1' from the first register?

- (c) (4 Points) What you observed in Problem 2b can be also achieved by a classical black box. However, what makes a maximally entangled state special is that you will obtain similar results if this time you perform measurement via the $\{|+\rangle, |-\rangle\}$ basis instead of the $\{|0\rangle, |1\rangle\}$. Namely, try the following circuit and see what you get:



- (d) (4 Points) There are three maximally entangled state in a two-qubit system that are orthogonal to $|\Phi^+\rangle$. They are called the *Bell states*. Can you prepare them?

Remark. Indeed, an arbitrary 2-qubit state (i.e. $|\psi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle$ with $|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1$) can be prepared from $|00\rangle$ by applying a sequence of 1-qubit gates and just a *single CX* gate. This justifies the significance of the *CX* gate. We are not going to prove this fact, but the interested students may find this in a theory course¹.

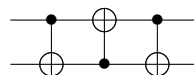
- (e) (4 Points) The SWAP gate (`circuit.swap(q[1],q[0])`)



exchanges the first register and the second register with respect to the computational basis.

Try to prepare a (not so naïve) 2-qubit state and verify the role of the swap gate.

Show that the SWAP gate is equivalent to the following circuit:



3. (20 points) **Global phase does not matter:** Suppose now we have two quantum states $|\psi_1\rangle = a|0\rangle + b|1\rangle$ and $|\psi_2\rangle = ac|0\rangle + bc|1\rangle$. Here, the vector forms of $|\psi_1\rangle$ and $|\psi_2\rangle$ are different from a *global factor* $c \in \mathbb{C}$, i.e. $|\psi_2\rangle = c|\psi_1\rangle$. In quantum mechanics, these two states $|\psi_1\rangle$ and $|\psi_2\rangle$ are considered to be *equivalent*; namely, the global phase $c \in \mathbb{C}$ does not matter².

In the following, you will get yourself a feeling about this fact via QISKit .

- (a) (4 points) The three basic Pauli logic gates are

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

¹In fact, any unitary operation on an n -qubit quantum system can be approximated to arbitrary accuracy by quantum circuits involved only the Hadamard gate, the *CX* gate, and the so-called *T* gate (they form a universal gate set); see e.g. Chapter 4.5 of Nielsen & Chuang's book.

²Here, you can verify that $|c| = 1$ (by using the facts that $|\psi_1\rangle$ and $|\psi_2\rangle$ are valid quantum states). Hence, one has $c = e^{i\theta}$ for some $\theta \in [0, 2\pi)$. That is why we call it a *phase* term.

Now prepare the following states $|\psi_1\rangle$ and $|\psi_2\rangle$:

$$|0\rangle \xrightarrow{H} \xrightarrow{X} |\psi_1\rangle$$

$$|0\rangle \xrightarrow{H} \xrightarrow{Y} \xrightarrow{Z} |\psi_2\rangle$$

What are the vector representations (with respect to the computational basis) of $|\psi_1\rangle$ and $|\psi_2\rangle$? Measure $|\psi_1\rangle$ and $|\psi_2\rangle$, respectively, and see what are the corresponding outcomes. Can you distinguish them (i.e. tell it is up or down circuit) from the measurement outcomes?

- (b) (10 points) Design arbitrary two states $|\psi_1\rangle$ and $|\psi_2\rangle$ that are different only up to a global phase by using whatever quantum gates you have learned from above. What are the states you tried? Demonstrate that they really are different only by a global phase (e.g. using `print(statevector)`). Measure them again and see the outcomes as you did in Problem 3a.

Remark. In fact, you cannot enumerate all possible states $|\psi_1\rangle$ and $|\psi_2\rangle$ that are different up to a global phase in your life time since the state space is a continuous set. However, in Problem 3b you have tried an *random* example. This should at least give you some confidence about this fact. This is one of the main purposes of doing numerical simulation. If you wisely design a simulation program and it also confirms your conjecture, then you are at a good position to analytically prove your claim. We will see this philosophy several times in this course.

- (c) (6 points) Now I hope that you have been convinced that global phase does not matter. How about the *relative phase*³? Try the following:

$$|0\rangle \xrightarrow{X} \xrightarrow{H} |\psi_1\rangle \xrightarrow{\text{Measurement}}$$

$$|0\rangle \xrightarrow{H} |\psi_2\rangle \xrightarrow{\text{Measurement}}$$

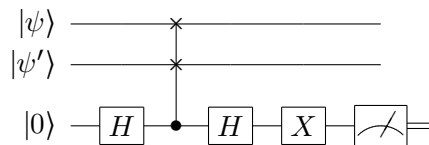
What are the two states $|\psi_1\rangle$ and $|\psi_2\rangle$ before measurement $\xrightarrow{\text{Measurement}}$? What are the measurement outcomes, respectively? Can we consider them to be *equivalent* as well? Why?

Remark. The reason that states up to a global phase are considered to be *physically equivalent* is because no one can physically tell the difference via *all* kinds of quantum measurements. However, you may wonder that they are still NOT *mathematically equivalent*. That's also true. Yet, remember that one of the main purposes of mathematics is to serve as a tool for modeling, analyzing, characterizing, and predicting Nature. If you couldn't make a difference physically, then why bother making things more complicated? Hence, in the mathematical languages describing quantum mechanics, we always treat the states $|\psi_1\rangle$ and $|\psi_2\rangle = c|\psi_1\rangle$ for $c \in \mathbb{C}$ as in an

³Taking computational basis $\{|0\rangle, |1\rangle\}$ as an example, for a state $|\psi\rangle = a|0\rangle + e^{i\theta}b|1\rangle$ for $a, b \in \mathbb{R}$, the term $\theta \in [0, 2\pi)$ or equivalently $e^{i\theta}$ is often referred to the *relative phase* between $|0\rangle$ and $|1\rangle$.

*equivalent class*⁴. That would potentially simplify the mathematical formalism without loss of generality. This is also an incidence of the *Principle of Occam's razor*.

4. (20 points) **The Swap test:** Suppose now both the first register is in some state $|\psi\rangle$ and the second register is in another state $|\psi'\rangle$. Please implement the following circuit to obtain the measurement outcome of the third register:



If the states $|\psi\rangle$ and $|\psi'\rangle$ that you prepared at the very beginning is actually the same, what do you get? If they are not the same, what now? What if the two states are orthogonal? If the two states are indeed the same (up to a global phase), can the Swap test tell you which states they are?

Hint: You may use the *Toffoli gate* (`ccx`) or the *controlled swap gate* (`cswap`).

5. (20 Points) **Quantum Random Number Generator (QRNG):** As you have seen, a quantum state is inherently random, i.e. each measurement outcome is random according to each certain probability amplitude. This fact could be annoying in practice. Suppose now that the desired computation outcome is, say '1001'; then it would be problematic if the measurement outcome being '1001' happens only with very small probability⁵.

However, the randomness of a quantum system might not be all that bad. Indeed we can exploit this fact to generate random numbers since the randomness is a resource; it's never cheap⁶. In this problem, your goal is to use n qubits (say, $n = 5$) to generate an n -bit random number between 0 and $2^n - 1$ (they are uniformly distributed).

- (a) (10 points) Design and build an n -bit random number generator using

```
simulator = Aer.get_backend('qasm_simulator').
```

What are the sequence of numbers you are generating? Are they random?

- (b) (10 points) In the course video, the Teaching Assistant has shown you how to use the real device from IBM Quantum Experience (note that you have to use your NTU account for registration). Now, run your quantum random number generator on an IBM real device and answer the above questions again?

Hint. If you have problem accessing the IBM real devices, watch the course video carefully, or you can read from https://qiskit.org/textbook/ch-labs/Lab01_QuantumCircui

⁴We are not going to explain what is an equivalent class in detail. But even in mathematics, you may have seen $0.999\ldots = 1 = 5/5 = 6 \bmod 5 = 3 \bmod 2$; and also in complexity theory: $5n^2 + 3n = O(n^2)$ & $3n^2 = O(n^2)$.

⁵Hence, one of the ultimate goals of a quantum algorithm is to make the probability amplitude of the desired result as large as possible, e.g. to make it almost orthogonal to other states.

⁶Nowadays, most random numbers are only *pseudo random*. How to generate *truly* random numbers is highly non-trivial.

[ts.html](#). Note that solving the issues (i.e. debugging) by digging into literature/specifications on your own is also important lesson you must take in an laboratory/experiment course.

(*Bonus 3 points*) There are many *pseudorandom number sequence test program*. You can test the random number you got to see whether they are really random or not. Why?

(*Bonus 5 points*) Can you use n qubits to generate more than n -bit (completely) random numbers? What is your reason?

3 Lab Report

There is no strict format/typesetting requirements for your lab report, but you have to make your report decent and looking nice. In the report, you should address the results of the exercises mentioned above. You should also include your simulation program in the appendix of the report. Include whatever discussions about the new findings during the lab exercise, or the problems encountered and how those are solved. Please properly cite the literature if you referred to. Do not limit yourself to the exercises specified here. You are highly encouraged to play around with your simulation program on self-initiated extra lab exercises/discussions. For example, you can see the [QISKit Textbook](#) and Chapter 4 of Nielsen & Chuang's book.

If you feel that quantum systems we have tried above is too abstract, no problem, there are many visualization tools in QISKit ; see e.g. https://qiskit.org/documentation/tutorials/circuits/2_plotting_data_in_qiskit.html. The lab exercises are good opportunities for you to make a quantum system more concrete or vivid by various simulation tools. You may plot all of the states you have created above for reification.

Besides doing the lab exercises, I have provided many of concepts, remarks, footnotes, discussions, thoughts, and my ideology. So you can have a flavor of thinking those stuff from my perspective. Of course, you may not agree with all of my points. (It is a good thing to have your own *critical thinking*). Nevertheless, digest all of the concepts you may find in the lab and provide your discussions in the report.

Apart from the QISKit , there are other SDKs for quantum program such as [Google's Cirq](#) and the [Microsoft Quantum Development Kit](#), and extension toolboxes such as [Xanadu's PennyLane](#) and the [TensorFlow Quantum](#); see e.g. <https://awesomeopensource.com/projects/quantum-computing>. For those who are interested, you are highly recommended to play with them in your final project.

For students who reckon Lab 1 as too simple or trivial, well, you certainly do not have to spend too much time on this Lab. Please go ahead to explore more interesting tasks via the resources given above or online. I would encourage you to start thinking about what final project you want to do and to start building upon it. Good luck!