Communication Systems Laboratory
Lab 1: Simulating Quantum Circuits with QISKit

(Report Due: 21:00, October 7, 2020)

1 Overview

The objective of this laboratory 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 will teach in Spring 2021). Due to the time constraint, we will only cover parts of quantum computing and classical/quantum communication in the course.

The focus of this course is different from the above-mentioned courses. Hence, those courses are not necessarily required. In this course, you will get your hands dirty 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 these systems.

QISKit is a software development kit (SDK) for quantum programming in the cloud and and is part of the IBM Quantum Experience cloud platform. The lab 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 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 five 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. **(15 points) Installing the QISKit :** QISKit is a package written in Python. Te access it, you must (i) have a Python platform in your computer; (ii) install the QISKit package; and (iii) create an *IBM Quantum Experience* account. Please follow the instructions given below and also read the QISKit Document to prepare yourself for joining the quantum information community.

   (a) Firstly, QISKit requires Python 3.5 or later. Please check that you have a Python Programming Platform, installed on your operating system. If not, we would recommend to use *Anaconda*. Download and install it.

   (b) Run

```
conda create −n IBMQ python=3 jupyter
```

(e.g. on your *Anaconda Prompt*) to create an environment 'IBMQ' that will be used to install the *qiskit* package.

Next, go to the environment

```
conda activate IBMQ
```

Now, we install the required package:

```
pip install qiskit matplotlib
```

Note that you have installed QISKit in the environment 'IBMQ'. Next time when you want to run QISKit , remember to go to this environment.

After installation, open your *Jupyter Notebook* by

```
jupyter notebook
```

Create a new *Python 3* in your Jupyter Notebook. Then `import qiskit` into your environment. If the QISKit was installed correctly, you can view the current version of QISKit by running `qiskit.__version__` .

(c) Before actually running QISKit , go to the website of *IBM Quantum Experience* to create a free account via your **NTU Account**[1]. Then, go to "My Account" to view your account setting and click on the "Copy token" to copy the token (e.g. say, `MY_API_TOKEN`) to your clipboard.

Now run the following commands to access the IBM Quantum Device:

```
from qiskit import IBMQ
apitoken = 'MY_API_TOKEN'
IBMQ.enable_account(apitoken)
```

(d) Now you are ready to play with QISKit . Before doing that, try running the following sample code to see if you got a nice histogram:

```
import qiskit
import numpy as np
from qiskit import QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram

# Create a Quantum Circuit acting on the q register
circuit = QuantumCircuit(2, 2)

# Add a H gate on qubit 0
circuit.h(0)

# Add a CX (CNOT) gate on control qubit 0 and target qubit 1
circuit.cx(0, 1)
```

[1]Throughout the Lab 1, you will only use QISKit 's simulator. However, if you would like to access IBM's real device, you have to the *NTU-IBM Quantum Hub* to apply the license via your IBM Quantum Experience account, which is only valid if you have an NTU Account.

```
    # Map the quantum measurement to the classical bits
    circuit.measure([0,1], [0,1])

    # Use Aer's qasm_simulator
    simulator = Aer.get_backend('qasm_simulator')

    # Execute the circuit on the qasm simulator
    job = execute(circuit, simulator, shots=10000)

    # Grab results from the job
    result = job.result()

    # Returns counts
    counts = result.get_counts(circuit)
    print("\nTotal count for 00 and 11 are:",counts)

    # Draw the circuit
    circuit.draw()
```

Please read the QISKit Document. Attach the figure of what you got (e.g. the quantum circuit diagram, results, and the histogram). Then you will receive the points of this problem. *Note:* If you have any questions, please read the QISKit document carefully, and you may find some information in https://www.youtube.com/c/qiskit/playlists.
For basic Python syntax, you can find, e.g. in the QISKit Textbook.

2. (**20 points**) **Manipulating a single qubit state:** In Problem 1, you have installed QISKit . Now in the present 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.

(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.

(c) Thirdly, using the registers created above to create a quantum circuit:

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

Now, the initial qubit of the circuit is in the state $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$. 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 calculate 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 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 $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$.

(d) Now we measure all the states with respect to the computational basis $\{|0\rangle, |1\rangle\}$, and use '`qasm_simulator`' in QISKit .Aeras 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, we should get each measurement outcome (stored in the classical registers) with certain probability that can be calculated by the *Born rule*.

(e) (4 points) Let us apply some single qubit gates on the initial qubit. Please apply the 'X' gate (i.e the *NOT gate*) $-\boxed{X}-$ 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 $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ 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 '$X \otimes I$':

$$\begin{array}{c} -\boxed{X}- \\ -\boxed{I}- \end{array}$$

where 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. 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 $-\boxed{H}-$ 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:

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

with a certain angle, say $\theta = \frac{\pi}{2}$ 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 U_3 gate (`circuit.u3()`):

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

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

(h) (8 points) Create a qubit state that will give a $\frac{2}{3}$ probability of measuring $|+\rangle$. 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 2h without using it.

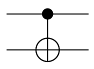
Remark. The interested students can refer to `https://qiskit.org/documentation/tutorials/circuits/3_summary_of_quantum_operations.html` finding the library of quantum operations available in the QISKit package.

3. **(15 points) Manipulating Multi-Qubit Gates:** In the previous problem, we have known how to apply a 1-qubit gate to prepare an arbitrary 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 simulate such a quantum circuit and quantum operation.

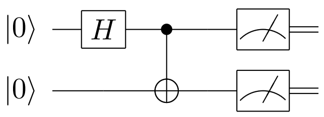
In this problem, you will learn how to use multi-qubit gates to prepare *entangled* quantum state, which makes your quantum computation much more profound.

(a) (4 Points) Apply the Hadamard gates $H \otimes H$ on the initial states $|0\rangle \otimes |0\rangle$ to see what you get. Now you may know how to prepare a superposition (with equal weight) of all n-qubit state aligning on the computational basis.

(b) (7 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;



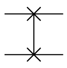
e.g. see the sample code given in Problem 1. Now, this state $\frac{|00\rangle+|11\rangle}{2}$ can not be expressed as tensor product of any two qubits. Hence, it is call an *entangled state*. In this case, the state $|\Phi^+\rangle := \frac{|00\rangle+|11\rangle}{2}$ is further called the *maximally entangled state*.

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

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?

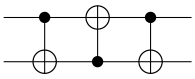
Remark. Indeed, an arbitrary 2-qubit state (i.e. $|\psi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle$ with $|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1$) can be prepared from $|00\rangle$ by applying a sequence of 1-qubit gates and just a *single* CX gate. This justifies the significance of the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

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

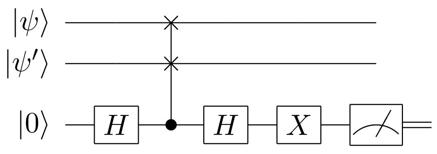


exchanges the first register and the second register with respect to the computational basis. Try to prepare a (not so naïve) 2-qubit state and verify the role of the swap gate.

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



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, can the Swap test tell you which states they are?

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

[2]In 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; see e.g. Section 4.5 of Nielsen and Chuang's book DOI:10.1017/cbo9780511976667.

5. (**30 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 probability[3].

   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 cheap[4] In this problem, your goal is to use n qubits (say, $n = 5$) to generate a 2^n-bit random number between 0 and $2^n - 1$ (they are uniformly distributed). This is then a striking application of quantum information science—somehow you have demonstrated that in this task a quantum system is *exponentially powerful* than the classical one. Please report your result and explain it. What is the pros and cons of a QRNG?

   (*Bonus 5 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?

   Hint. The two important quantum resources are the *superposition* and the *entanglement*. You may need one of them. Since the measurement is respect to the computational basis, you may get 2^n different outcomes. Try, e.g. 1024 shots, and set a number as threshold. Some of them might be smaller than the threshold or higher than that. Flip them to either '0' or '1' so you get a 2^n-bit number.

3 Lab Report

There is no 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 are those 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.

Besides of 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.

[3]Hence, 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.

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