QUANTUM PROGRAMMING | HOMEWORK ONE
BENJAMIN NGUYEN

Assignment due on Canvas by Friday, January 31 at 23:59

Problem 1

(a) Prove that the tensor product is bilinear: for all $\alpha \in \mathbb{C}$, all $|\psi_1\rangle, |\psi_2\rangle \in \mathcal{H}_A$, and all $|\phi_1\rangle, |\phi_2\rangle \in \mathcal{H}_B$, the following three conditions hold:

- $\alpha(|\psi_1\rangle \otimes |\phi_1\rangle) = (\alpha|\psi_1\rangle) \otimes |\phi_1\rangle = |\psi_1\rangle \otimes (\alpha|\phi_1\rangle)$,
- $(|\psi_1\rangle + |\psi_2\rangle) \otimes |\phi_1\rangle = |\psi_1\rangle \otimes |\phi_1\rangle + |\psi_2\rangle \otimes |\phi_1\rangle$,
- $|\psi_1\rangle \otimes (|\phi_1\rangle + |\phi_2\rangle) = |\psi_1\rangle \otimes |\phi_1\rangle + |\psi_1\rangle \otimes |\phi_2\rangle$.

(b) Prove that for all product states $|\psi_1\rangle \otimes |\phi_1\rangle, |\psi_2\rangle \otimes |\phi_2\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$, it holds that

$$(\langle\psi_1| \otimes \langle\phi_1|)(|\psi_2\rangle \otimes |\phi_2\rangle) = \langle\psi_1|\psi_2\rangle \cdot \langle\phi_1|\phi_2\rangle.$$

(c) Prove that the norm of a tensor product is the product of the norms, i.e., prove that for all $|\psi\rangle \in \mathcal{H}_A$ and all $|\phi\rangle \in \mathcal{H}_B$,

$$\||\psi\rangle \otimes |\phi\rangle\| = \||\psi\rangle\| \cdot \||\phi\rangle\|.$$

Solution

(a) Definition of Bilinear: Let V, W, X be three vector spaces. A *bilinear map* from $V \times W$ to X is a function $H : V \times W \to X$ such that:

$$H(av_1 + v_2, w) = aH(v_1, w) + H(v_2, w) \quad \text{for } v_1, v_2 \in V, w \in W, a \in \mathbb{F}$$

$$H(v, aw_1 + w_2) = aH(v, w_1) + H(v, w_2) \quad \text{for } v \in V, w_1, w_2 \in W, a \in \mathbb{F}$$

These are simply the defining properties of the tensor product as a bilinear map

$$(\,\cdot\,,\,\cdot\,) : \mathcal{H}_A \times \mathcal{H}_B \;\to\; \mathcal{H}_A \otimes \mathcal{H}_B.$$

By definition, the tensor product is linear in each slot separately, which exactly implies all three properties above. $\square$

(b) Choose orthonormal bases $\{|e_i\rangle\}_{i=1}^n$ for $\mathcal{H}_A$ and $\{|f_j\rangle\}_{j=1}^m$ for $\mathcal{H}_B$. Expand:

$$|\psi_1\rangle = \sum_{i=1}^{n} a_i\,|e_i\rangle, \quad |\phi_1\rangle = \sum_{r=1}^{m} b_r\,|f_r\rangle,$$

so

$$\langle\psi_1| = \sum_{i=1}^{n} a_i^*\langle e_i|, \quad \langle\phi_1| = \sum_{r=1}^{m} b_r^*\langle f_r|.$$

Thus,

$$(\langle\psi_1|\otimes\langle\phi_1|) = \Big(\sum_{i=1}^{n} a_i^*\langle e_i|\Big) \otimes \Big(\sum_{r=1}^{m} b_r^*\langle f_r|\Big) = \sum_{i=1}^{n}\sum_{r=1}^{m} a_i^*\, b_r^*\, \langle e_i|\otimes\langle f_r|.$$

Similarly, expand

$$|\psi_2\rangle = \sum_{j=1}^{n} c_j\,|e_j\rangle, \quad |\phi_2\rangle = \sum_{s=1}^{m} d_s\,|f_s\rangle$$

to get

$$|\psi_2\rangle\otimes|\phi_2\rangle = \sum_{j=1}^{n}\sum_{s=1}^{m} c_j\, d_s\, |e_j\rangle\otimes|f_s\rangle.$$

Applying one to the other:

$$(\langle\psi_1|\otimes\langle\phi_1|)\,(|\psi_2\rangle\otimes|\phi_2\rangle) = \sum_{i,j}\sum_{r,s} a_i^*\, b_r^*\, c_j\, d_s \big(\langle e_i|\otimes\langle f_r|\big)\big(|e_j\rangle\otimes|f_s\rangle\big).$$

By definition of the tensor-product inner product,

$$\langle e_i|\otimes\langle f_r|\,\big(|e_j\rangle\otimes|f_s\rangle\big) = \langle e_i|e_j\rangle\,\langle f_r|f_s\rangle = \delta_{ij}\,\delta_{rs}.$$

Hence the sums collapse to

$$\sum_{i=1}^{n}\sum_{r=1}^{m} a_i^*\, b_r^*\, c_i\, d_r = \Big(\sum_{i=1}^{n} a_i^*\, c_i\Big)\Big(\sum_{r=1}^{m} b_r^*\, d_r\Big) = \langle\psi_1|\psi_2\rangle\,\langle\phi_1|\phi_2\rangle,$$

as claimed. □

(c) By definition,

$$\big\|\,|\psi\rangle\otimes|\phi\rangle\big\|^2 = \big\langle\psi\otimes\phi\;\big|\;\psi\otimes\phi\big\rangle = \langle\psi|\psi\rangle\,\langle\phi|\phi\rangle = \|\psi\|^2\,\|\phi\|^2.$$

Therefore: $\|\psi\otimes\phi\| \;=\; \|\psi\|\,\|\phi\|$. □

Problem 2

(a) Prove the *no-cloning theorem*: For all $N > 1$, there does *not* exist $U \in \mathrm{U}(N^2)$ such that for all $|\psi\rangle, |\phi\rangle \in \mathbb{C}^N$,[1]

$$U|\psi\rangle|\phi\rangle = |\psi\rangle|\psi\rangle.$$

(b) How does this differ from classical computation? (*Hint:* Think of the case where the quantum states are qubits. Can you clone qubits? Can you clone bits?)

(c) Prove that for all $N > 1$ and all $|\psi\rangle, |\phi\rangle \in \mathbb{C}^N$, there exists $U \in \mathrm{U}(N^2)$ such that

$$U|\psi\rangle|\phi\rangle = |\psi\rangle|\psi\rangle.$$

(d) Explain why this does not violate the no-cloning theorem.

(e) Argue that (c) and the no-cloning theorem imply that experimental physicists can create many copies of a *known* quantum state but not an *arbitrary* quantum state.

(a) Assume for contradiction that such a universal cloner U exists. Then, in particular,

$$U\big(|\psi\rangle \otimes |\phi\rangle\big) \;=\; |\psi\rangle \otimes |\psi\rangle \quad \text{and} \quad U\big(|\phi\rangle \otimes |\phi\rangle\big) \;=\; |\phi\rangle \otimes |\phi\rangle.$$

Because U is unitary, it preserves inner products. Consider the inner products

$$\big(\langle\psi|\phi\rangle\big)\big(\langle\phi|\phi\rangle\big) \;=\; \langle\psi|\phi\rangle,$$

since $\langle\phi|\phi\rangle = 1$.

$$\big(\langle\psi|\phi\rangle\big)\big(\langle\psi|\phi\rangle\big) \;=\; |\langle\psi|\phi\rangle|^2.$$

Equating these (by unitarity) gives

$$\langle\psi|\phi\rangle \;=\; \big|\langle\psi|\phi\rangle\big|^2.$$

The only real solutions to $x = x^2$ are $x = 0$ or $x = 1$. Hence $|\psi\rangle$ and $|\phi\rangle$ must be either orthogonal or identical. But we assumed $|\psi\rangle$ and $|\phi\rangle$ were arbitrary, and non-orthogonal states do exist. This contradiction shows no such universal U can exist. $\square$

(b) Difference from classical computation

Classical bits 0 and 1 can be copied: $b \mapsto (b, b)$. There is no obstruction to duplicating any unknown bit. In contrast, quantum states can be superpositions; the linearity of quantum mechanics and the requirement of preserving inner products imply that no single unitary can clone all states.

(c) Existence of a Cloner for Specific States It is possible to clone one particular pair $\big(|\psi\rangle, |\phi\rangle\big)$ using an appropriate unitary. For any $N > 1$ and any fixed $|\psi\rangle, |\phi\rangle \in \mathbb{C}^N$, there exists a unitary $U \in U(N^2)$ such that

$$U\big(|\psi\rangle \otimes |\phi\rangle\big) \;=\; |\psi\rangle \otimes |\psi\rangle.$$

[1] Notice, whereas in the last question I used the tensor product symbol $\otimes$, here I am deliberately choosing not to so you can get used to both conventions.

- Extend $\{|\psi\rangle, |\phi\rangle\}$ to an orthonormal basis of $\mathbb{C}^N$.
- Define U to map $|\psi\rangle \otimes |\phi\rangle \mapsto |\psi\rangle \otimes |\psi\rangle$, and continue this definition on an orthonormal basis for $\mathbb{C}^N \otimes \mathbb{C}^N$ in any consistent, unitary way.

Since we only require U to clone one particular pair, no contradiction arises.

(d) The no-cloning theorem forbids a universal cloner. Part (c) constructs a unitary that works only for one specific $\big(|\psi\rangle, |\phi\rangle\big)$. If we present a different state, it will fail to clone. Hence there is no contradiction with the theorem.

(e) The no-cloning theorem show that we can clone a state if we already know its description. Once you we precisely which state $|\psi\rangle$ is, we can simply build or prepare as many copies of $|\psi\rangle$ as we like. However, we cannot universally clone an arbitrary, unknown quantum state. This crucial distinction between known and unknown states underlies the fundamental impossibility result of no-cloning.

Problem 3

This problem concerns the following matrix (written in the computational basis), which is called the *Hadamard gate*:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

In this course, we will see that H is one of the most fundamental matrices in quantum computation, as it appears in almost every interesting quantum algorithm we know. For this reason, let us learn a bit about it.

(a) Prove that H is Hermitian.

(b) Prove that $H \in \mathrm{U}(2)$. Conclude that H is an involution.

(c) Using the Bloch sphere, explain geometrically how H acts on the computational basis states $|0\rangle$ and $|1\rangle$.

(d) Let $H^{\otimes n}$ denote the n-fold tensor product $H \otimes H \otimes \cdots \otimes H$ and let 0^n denote the n-bit, all-zero string $00\ldots0$. Prove that the probability distribution one obtains by measuring the state $H^{\otimes n}|0^n\rangle$ in the computational basis is the uniform distribution over $\{0,1\}^n$. In other words, prove that for all $y \in \{0,1\}^n$, $|\langle y|H^{\otimes n}|0^n\rangle|^2 = 2^{-n}$.

(a) A matrix M is Hermitian if $M^\dagger = M$. Since H is a real matrix, its conjugate transpose is just its transpose. So

$$H^\dagger = \left(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\right)^\dagger = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^T = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = H.$$

Thus H is Hermitian.

(b) A matrix M is unitary if $M^\dagger M = I$. We already have $H^\dagger = H$, so

$$H^\dagger H = HH = \Big(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\Big)\Big(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\Big)$$

Therefore

$$H^\dagger H = \frac{1}{2} \begin{pmatrix} 1+1 & 1-1 \\ 1-1 & 1+1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I.$$

Hence $H \in U(2)$. Also $H^2 = I$ implies H is its own inverse, so H is called an involution.

(c) In the computational basis $\{|0\rangle, |1\rangle\}$, we have

$$H|0\rangle = \frac{1}{\sqrt{2}} \big(|0\rangle + |1\rangle\big), \quad H|1\rangle = \frac{1}{\sqrt{2}} \big(|0\rangle - |1\rangle\big).$$

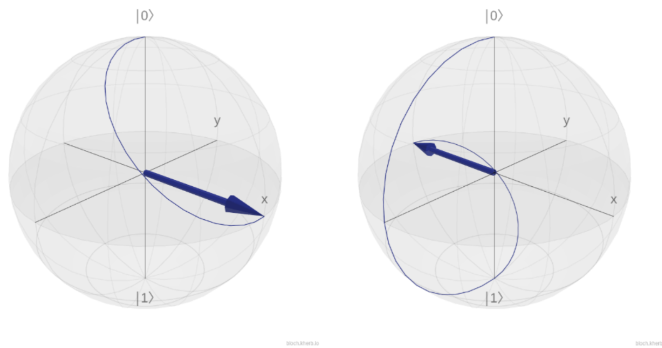


Figure 1: The Bloch sphere (link to visualizer) after applying H gate for computational basis $\{|0\rangle, |1\rangle\}$. The $|1\rangle$ is constructed by using X gate to flip $|0\rangle$.

Viewed on the Bloch sphere: $|0\rangle$ is mapped to $|+\rangle$, and $|1\rangle$ goes to $|-\rangle$. Thus H sends the z-axis of the Bloch sphere to the x-axis.

(d) Let $|0^n\rangle$ be the n-qubit all-zero state. Then

$$H^{\otimes n} |0^n\rangle = \Big(H|0\rangle\Big)^{\otimes n} = \Big(\frac{1}{\sqrt{2}} (|0\rangle + |1\rangle)\Big)^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle.$$

This is an equal superposition over all 2^n computational-basis states. Measuring in the computational basis, the probability of each outcome $|y\rangle$ is

$$\big|\langle y \mid H^{\otimes n} |0^n\rangle\big|^2 = \left|\frac{1}{\sqrt{2^n}}\right|^2 = \frac{1}{2^n},$$

i.e. the outcomes are uniformly distributed over $\{0,1\}^n$.

Problem 4 (Writing)

Standard textbooks on quantum mechanics say that there are two ways in which a quantum state vector $|\psi\rangle$ can evolve. The first way is unitarily (a.k.a. evolution according to the Schrödinger equation), the second way is non-unitarily (a.k.a. evolution resulting from a *measurement*). However, it is never precisely said *when* a quantum state is evolving unitarily versus when it is evolving non-unitarily, except something along the lines of "it evolves non-unitarily if and only if the system is being measured", for some imprecise and fuzzy word "measured". This is the germ of (one part of) the *measurement problem in quantum mechanics*, which is a very polarizing thing in physics circles.[2]

Write a paragraph that briefly addresses the following questions: Given the profound success of quantum mechanics in explaining the universe, do you think it's important to resolve the measurement problem? Why or why not? Is it obvious when a quantum state is being measured (and hence evolving non-unitarily) versus when it is not (and hence evolving unitarily)?

The measurement problem highlights the tension between unitary evolution, as prescribed by the Schrödinger equation, and the apparent non-unitary "collapse" when a system is measured. While quantum mechanics is extraordinarily successful in predicting experimental outcomes, it leaves open the exact criteria for when and how measurement-induced collapse occurs. Many physicists find it crucial to resolve this issue to achieve a fully coherent theory of quantum reality, but others argue that the existing framework suffices for all practical purposes. In practice, it is not always obvious whether a system is "being measured" or simply interacting with another quantum system, since the dividing line between "measuring apparatus" and "measured object" can be ambiguous. Consequently, the transition from unitary to non-unitary evolution, though robustly observed, remains conceptually elusive.

[2]For those interested in this and the philosophy of quantum physics more generally, I recommend Adam Becker's great book, *What is Real?: The Unfinished Quest for the Meaning of Quantum Physics.*

Problem 5 (Programming)

In this problem, you will make a 4-bit quantum random number generator in Qiskit. The following is a step-by-step guide to getting Qiskit up and running. By part (v), you should have everything you need to build your random number generator.

(i) Install Qiskit on your machine by following IBM's Qiskit Installation Guide at:[3]

https://docs.quantum.ibm.com/guides/install-qiskit

Note, it is recommended that you use Python virtual environments so that Qiskit is separated from other applications. Also, we will not be running jobs on actual quantum hardware, so you do *not* need to install `qiskit-ibm-runtime` in Step 3. Step 4 is also optional.

(ii) In your virtual environment, install `qiskit-aer` by running

```
pip install qiskit-aer
```

This installs a vast suite of quantum computer simulators known as Qiskit Aer.

(iii) With Qiskit installed, open a new file in your favorite text editor and name it "YOURLASTNAME_HW1.py".

(iv) As mentioned, the goal of this problem is to make a quantum circuit that acts as a 4-bit random number generator. I will help you get started by implementing a different circuit.

In your .py file, insert:

```
from qiskit import *
qr = QuantumRegister(2,'q')
cr = ClassicalRegister(2, 'c')
```

You know what the first line means. In the second line, `qr` is a *quantum register*, which is just a collection of qubits whose overall state has not yet been specified. The command `QuantumRegister(2,'q')` tells Qiskit to construct a 2 qubit register, and to name the register '`q`'. Similarly, `cr` is a classical register, which is just a collection of bits whose values are not yet specified. The command `ClassicalRegister(2,'c')` tells Qiskit to construct a 2-bit register, and to name the register '`c`'. We need a classical register because that is where the measurement outcomes of the qubits will be stored.

To see what this all looks like, insert the following two lines and then run the file:[4]

```
qc = QuantumCircuit(qr,cr)
print(qc)
```

This makes a (rather trivial) *quantum circuit* with quantum register `qr` and classical register `cr`. As output, you should get something that looks like:

```
q_0:
q_1:
c: 2/
```

[3] If you ever need to know anything at all about Qiskit, it is documented here.

[4] There are nicer ways to view the circuit besides `print`. If you're interested, look here.

Here, `q_0` and `q_1` constitute the quantum register `qr`, and together represent individual qubits that are initialized in the state $|0\rangle$. (When we call the `QuantumCircuit` class, the qubits in the quantum register are all initialized to $|0\rangle$.) Also, the `2/` in the classical register indicates that it holds 2 bits, so in particular we can write 2 bits worth of information from the quantum register to the classical register by measuring the quantum register.

To prove to you that the state of each qubit is $|0\rangle$, and hence that the state of the two qubits is $|0\rangle \otimes |0\rangle$, insert the following:

```
from qiskit.quantum_info import Statevector
state = Statevector(qc)
print(state.data)
```

The first line is necessary because subpackages of Qiskit need to be imported separately. The next line calls `Statevector(qc)`, which asks Qiskit to construct the quantum state at the end of the circuit `qc`. The `data` attribute of `state` is the quantum state. Printing, you should get the following as output:

```
[1.+0.j 0.+0.j 0.+0.j 0.+0.j]
```

Indeed, up to transposition, this is $|0\rangle \otimes |0\rangle$ written in the computational basis (recall that in Python a number like `1.` denotes a floating point number and `j` denotes $\sqrt{-1}$).

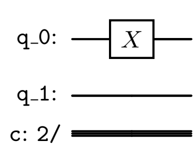
Now, let's make our circuit do something more than just the 4×4 identity gate I_4 (is it clear to you why `qc` is implementing I_4?). First, suppose we want our quantum circuit `qc` to put the first qubit in the state $|1\rangle$. To do this, we need to apply $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ to `q_0`. Indeed, you can check that $X|0\rangle = |1\rangle$. At the end of the day, we then expect the overall state of the two qubits to be the $\mathbb{C}^4$ computational basis state

$$\begin{aligned}(X \otimes I_2)|0\rangle \otimes |0\rangle &= |1\rangle \otimes |0\rangle \\ &= \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}.\end{aligned}$$

To do this in Qiskit, first remove or comment out the `state` from before, and replace it with

```
qc.x(0)
state = Statevector(qc)
```

Here, the first line says, "to `qc`, append an X gate to qubit `q_0`". In general, `qc.x(i)` applies an X gate to qubit `q_i`. If you now run `print(qc)`, you should see that indeed we are applying an X gate to qubit `q_0`:



Therefore, this sure seems like what we want to be doing. Let's see if it is by running `print(state.data)`. You should get

```
[0.+0.j 1.+0.j 0.+0.j 0.+0.j]
```

This is *almost* right, except that it corresponds to the computational basis state $|0\rangle \otimes |1\rangle$ and not $|1\rangle \otimes |0\rangle$, which is what we want! Alas, we have stumbled upon an annoying convention of Qiskit. Contrary to how we (and many others, like Nielsen and Chuang) talk about the order of qubits in a circuit—namely, by placing the qubit on the top line of the quantum circuit (`q_0`) in the left-most position in the ket, the qubit on the second-to-top line of the circuit (`q_1`) to the second-left-most position in the ket, and so on, so that the state for `qc` is $|$`q_0q_1`$\rangle$—Qiskit uses the reverse convention, and places the qubit on the top line (`q_0`) in the *right*-most position of the ket, the qubit on the second-to-top line (`q_1`) to the second-*right*-most position in the ket, and so on, so that to Qiskit the state of `qc` is actually $|$`q_1q_0`$\rangle$. To learn more about this, click here (it is related to the convention of whether in a bit string $q_0 q_1 \dots q_n$ you regard q_0 as the least or most significant bit).

To force Qiskit to use our convention (in which q_0 is the most significant bit), we can force `qc` to use a temporary "reverse ordering convention", and then compute the state vector:

```
state = Statevector(qc.reverse_bits())
print(state.data)
```

Here, the output is

```
[0.+0.j 0.+0.j 1.+0.j 0.+0.j]
```

which is the matrix form of $|1\rangle \otimes |0\rangle$, according to our convention. Take note that `reverse_bits()` literally swaps `q_0` and `q_1` in the quantum circuit `qc`, so if you try to do something sneaky and permanently *redefine* `qc` as `qc = qc.reverse_bits()`, then the circuit is going to look reversed when you print it. I discourage such permanent redefining because this reversal can quickly get confusing.

Now, let us make a slightly more complicated circuit. In addition to putting the first qubit `q_0` into state $|1\rangle$, let us now put the second qubit `q_1` into the superposition state $|+\rangle := \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. To do this, we need to apply the Hadamard gate H to `q_1`. Indeed, you can check that

$$
\begin{aligned}
(X \otimes H)|0\rangle \otimes |0\rangle &= |1\rangle \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \\
&= \frac{1}{\sqrt{2}}\left(|1\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle\right) \\
&= \begin{pmatrix} 0 \\ 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}.
\end{aligned}
$$

To do this in Qiskit, right after the `qc.x(0)` command from above, put

```
qc.h(1)
```

This says, "to `qc`, append an H gate to qubit `q_1`". Again running

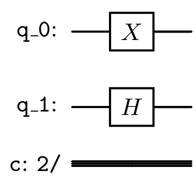
```
state = Statevector(qc.reverse_bits())
print(state.data)
```

you should get

```
[0.+0.j 0.+0.j 0.70710678+0.j 0.70710678+0.j]
```

Since $\frac{1}{\sqrt{2}} \approx 0.707106$, this agrees with the state above.

If you want to be doubly sure that this circuit is indeed applying an X gate to qubit `q_0` and an H gate to qubit `q_1`, you can reprint the circuit by running `print(qc)`. It should look something like:



(If you had permanently redefined `qc` as `qc = qc.reverse_bits()`, which again I discourage, then you will not get the above result, but rather a reversed version.)

At this point, we have a quantum circuit which maps $|0\rangle \otimes |0\rangle$ to $|1\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Let us now see how to measure the output state, and write the measurement result to the classical register.

For a single computational basis measurement of both qubits, simply insert:

```
qc.measure(0,0)
qc.measure(1,1)
```

This tells Qiskit to measure qubit `q_0` in the computational basis, and record the result (which is either 0 or 1) into the first bit of the classical register, and then to measure qubit `q_1` in the computational basis, and record the result into the second bit of the classical register. In general, `qc.measure(i,j)` tells Qiskit to measure qubit `q_i` in the computational basis, and record the result into the (`j` + 1)th bit of the classical register. You should print the quantum circuit again to see how adding a measurement changes the circuit.

Now is the time to actually *run* the circuit and get a 2-bit output. To do this, we will use one of the most basic simulators in Qiskit Aer, which is a vast suite of quantum circuit simulators. In particular, we will use the `QasmSimulator`. To use this simulator, import it using

```
from qiskit_aer import QasmSimulator
```

Finally, to run the circuit, say 100 times, write

```
backend = QasmSimulator()
result = backend.run(qc.reverse_bits(), shots=100).result()
counts = result.get_counts()
```

Here, `backend` specifies that we are using the Qiskit Aer `QasmSimulator` to simulate our quantum circuit, and `backend.run(qc.reverse_bits(), shots=100)` simulates our circuit `qc` 100 times, and uses our reverse ordering convention. Use `.result()` to get the data from the 100 trials, and additionally call `.get_counts()` on `result` to put the data into a dictionary. You should `print(count)` to understand what the output looks like.

You are now ready to build a 4-bit quantum random number generator. You should comment out or delete all the above code, and start fresh for the final, unguided part of this problem.

(v) In Qiskit, build a 4-bit quantum random number generator, i.e., a quantum circuit with a 4-qubit register (that writes into a classical 4-bit register) that on input $|0^4\rangle = |0\rangle^{\otimes 4}$ outputs $y \in \{0,1\}^4$ with probability 2^{-4} when measured in the computational basis. (*Hint: recall problem 3 above!*). Using Matplotlib or some other Python graphing software, output a histogram of 10 000 trials of your circuit to convince me that it is a random number generator. The x-axis should contain the 16 4-bit strings 0000, 0001, etc., and the y-axis should specify the number of times a given string is measured.

(vi) Submit to Canvas both your .py file and a picture of your histogram.

Please see the `Results.png` file and `NGUYEN_HW1.py` for the solution.