SCS 43XX

Quantum Mechanics in Computing

University of Colombo, School of Computing

Labsheet 01

Introduction to Qiskit

In this lab, you will get an introduction to Qiskit, the open-source quantum computing software development framework. Before diving into the quantum states, ensure that Qiskit is installed and set up on your system.

1. Install Qiskit: To begin, you need to install Qiskit. You can follow the instructions on the official Qiskit website or use the command below to install it via pip:

```
pip install qiskit
```

For a step-by-step guide, you can watch this video tutorial:
Click here to watch the tutorial on installing Qiskit.

2. Initialize the following states in Qiskit:

1. $|u\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
2. $|v\rangle = \frac{1+2i}{3}|0\rangle - \frac{2}{3}|1\rangle$
3. $|w\rangle = \frac{1}{3}|0\rangle + \frac{2}{3}|1\rangle$

Answer: Below is the graphical representation of the quantum state vectors u, v, and w:

```python
from qiskit.quantum_info import Statevector
from numpy import sqrt

u = Statevector([1 / sqrt(2), 1 / sqrt(2)])
v = Statevector([(1 + 2.0j) / 3, -2 / 3])
w = Statevector([1 / 3, 2 / 3])

print("State vectors u, v, and w have been defined.")
```

Output:

```
State vectors u, v, and w have been defined.
```

Figure 1: Graphical representation of the state vectors u, v, and w.

3. Print the States u, v, and w using Qiskit: Use the Qiskit `draw()` method to display the quantum states u, v, and w.

Answer: Below is the output of the `draw()` method:

```python
display(u.draw("latex"))
display(v.draw("text"))
```

Output:

$$\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$

```
[ 0.33333333+0.66666667j,-0.66666667+0.j        ]
```

Figure 2: Graphical output of the quantum states u, v, and w.

4. Verify if the Statevectors u, v, and w are valid quantum states using the is_valid method:
The `Statevector` class in Qiskit includes the `is_valid` method, which checks if a given vector is a valid quantum state vector. A valid quantum state vector must satisfy the Euclidean norm condition $||\psi|| = 1$.

Answer: Below is the Python code used to verify the validity of the quantum state vectors:

```python
display(u.is_valid())
display(w.is_valid())
```

Output:

```
True

False
```

Figure 3: Output showing the validity of the state vectors u, v, and w.

5. One way to measure

Next we will see one way that measurements of quantum states can be simulated in Qiskit, using the `measure` method from the `Statevector` class.

Code cells can be modified — so go ahead and change the specification of the vector if you wish.

Next, running the `measure` method simulates a standard basis measurement. It returns the result of that measurement, plus the new quantum state of our system after that measurement.

Answer: Below is the Python code used to verify the validity of the quantum state vectors:

```
v.measure()
```

Output:

```
('1',
 Statevector([ 0.+0.j, -1.+0.j],
             dims=(2,)))
```

Figure 4: Output showing the validity of the state vectors u, v, and w.

6. Measurement Outcomes

Quantum measurements are inherently probabilistic, meaning the same method can produce different results on repeated trials. Run the `measure` method multiple times to observe this variability.

For the specific case of the vector v, the `measure` method defines the quantum state vector after the measurement. Depending on the measurement outcome:

- If the outcome is 0, the resulting state may appear as $\frac{1+2i}{5}|0\rangle$, rather than $|0\rangle$.
- If the outcome is 1, the resulting state may appear as $-|1\rangle$, rather than $|1\rangle$.

These states are equivalent as they differ only by a global phase—a complex number with unit magnitude. For now, this distinction can be ignored and will be explained further in Lesson 3.

Error Handling with Invalid Quantum States

The `Statevector` class will raise an error if the `measure` method is applied to an invalid quantum state vector. This behavior ensures the integrity of quantum operations. You can experiment with invalid states to observe the error output.

Simulating Multiple Measurements

The `sample_counts` method from the `Statevector` class allows for simulating repeated measurements. For example, measuring the vector v 1000 times will, with high probability, result in:

- Outcome 0: Approximately $\frac{5}{9}$ of the trials (556 out of 1000).

- Outcome 1: Approximately $\frac{4}{9}$ of the trials (444 out of 1000).

The code below demonstrates this simulation and uses the `plot_histogram` function for visualization:

```
from qiskit.visualization import plot_histogram

statistics = v.sample_counts(1000)
display(statistics)
plot_histogram(statistics)
```

Output:

```
{'0': 554, '1': 446}
```

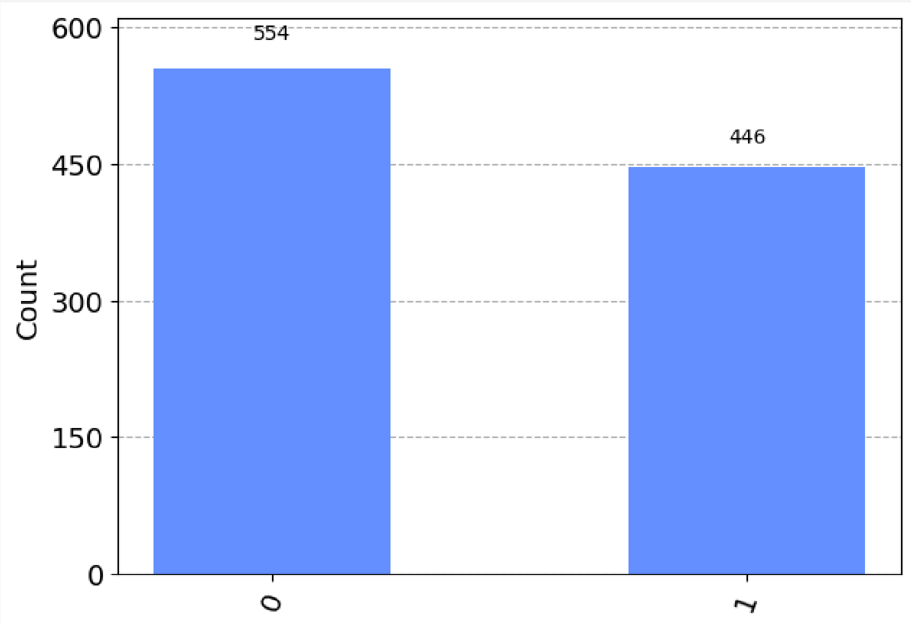

Figure 5: Histogram of measurement outcomes over 1000 trials.

By varying the quantum state vector or the number of trials, you can explore how measurement probabilities align with theoretical predictions.

8.Quantum circuit with a single qubit

Experiment with composing qubit unitary operations using Qiskit's `QuantumCircuit` class. In particular, we may define a quantum circuit (which in this case will simply be a sequence of unitary operations performed on a single qubit) as follows.

Answer: Below is the Python code used for creating a one-qubit basic circuit

```python
from qiskit import QuantumCircuit

circuit = QuantumCircuit(1)

circuit.h(0)
circuit.t(0)
circuit.h(0)
circuit.t(0)
circuit.z(0)

circuit.draw()
```

Output:

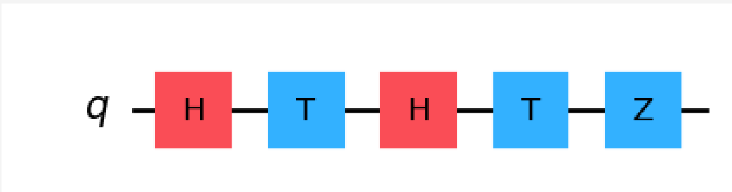

Figure 6: basic circuit

9. Quantum evolution of a single system
The operations are applied sequentially, starting on the left and ending on the right in the figure. Let us first initialize a starting quantum state vector and then evolve that state according to the sequence of operations.

Answer: Below is the Python code used for applying the circuit and printing the state

```python
ket0 = Statevector([1, 0])
v = ket0.evolve(circuit)
v.draw("text")
```

Output:

```
[ 0.85355339+0.35355339j,-0.35355339+0.14644661j]
```

Figure 7: quantum evolution

10. Measure 4000 times

Finally, let's simulate the result of running this experiment (i.e., preparing the state $|0\rangle$, applying the sequence of operations represented by the circuit, and measuring) 4000 times.

Answer:

```
statistics = v.sample_counts(4000)
plot_histogram(statistics)
```

Output:

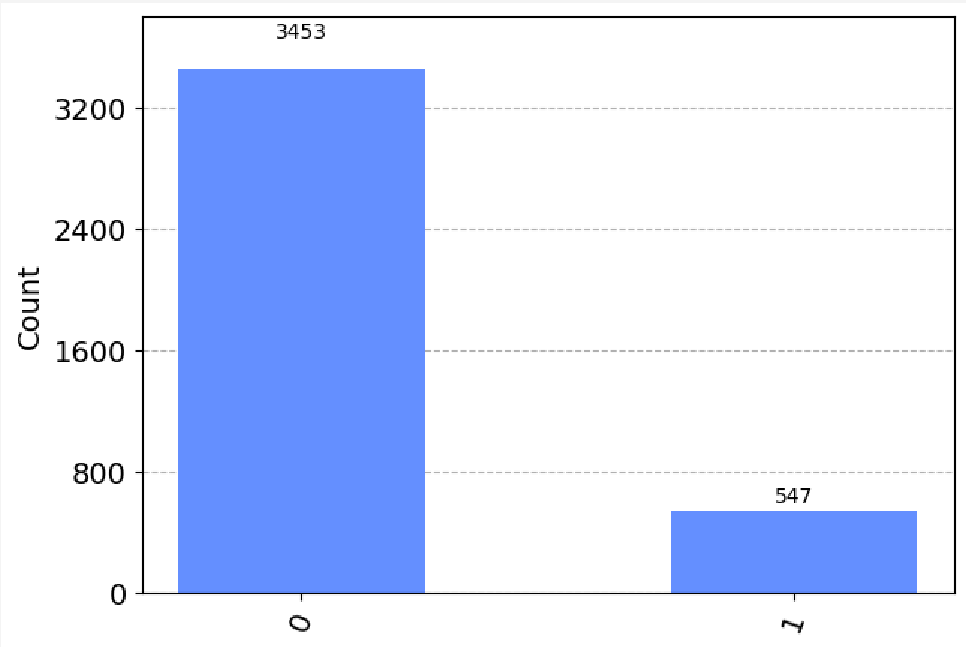

Figure 8: quantum evolution