**Qiskit simulator**

Qiskit Aer is our package for simulating quantum circuits. It provides many different backends for doing a simulation.

**Statevector backend**

The most common backend in Qiskit Aer is the statevector\_simulator. This simulator returns the quantum state, which is a complex vector of dimensions 2 ^ n (2 to the power n), where n is the number of qubits.

-- Start of code ---  
# Import Aer

from qiskit import Aer

# Run the quantum circuit on a statevector simulator backend

backend = Aer.get\_backend('statevector\_simulator')

In Qiskit, the execute function is used to compile and run the quantum circuit. execute returns a job object that encapsulates information about the job submitted to the backend.

# Create a Quantum Program for execution

job = execute(circ, backend)

When you run a program, a job object is made that has the following two useful methods: job.status() and job.result(), which return the status of the job and a result object, respectively.

Jobs run asynchronously, but when the result method is called, it switches to synchronous and waits for it to finish before moving on to another task.

result = job.result()

The results object contains the data and Qiskit provides the method result.get\_statevector(circ) to return the state vector for the quantum circuit.

outputstate = result.get\_statevector(circ, decimals=3)

print(outputstate)

Qiskit also provides a visualization toolbox to allow you to view these results.

Below, we use the visualization function to plot the real and imaginary components of the state density matrix ρ.

from qiskit.visualization import plot\_state\_city

plot\_state\_city(outputstate)

**Unitary backend**

Qiskit Aer also includes a unitary\_simulator that works provided all the elements in the circuit are unitary operations. This backend calculates the 2n×2n matrix representing the gates in the quantum circuit.

# Run the quantum circuit on a unitary simulator backend

backend = Aer.get\_backend('unitary\_simulator')

job = execute(circ, backend)

result = job.result()

# Show the results

print(result.get\_unitary(circ, decimals=3))

**OpenQASM backend¶**

The simulators above are useful because they provide information about the state output by the ideal circuit and the matrix representation of the circuit. However, a real experiment terminates by measuring each qubit (usually in the computational |0⟩,|1⟩ basis). Without measurement, we cannot gain information about the state. Measurements cause the quantum system to collapse into classical bits.

For example, suppose we make independent measurements on each qubit of the three-qubit GHZ state

|ψ⟩=(|000⟩+|111⟩)/√2,

and let xyz denote the bitstring that results. Recall that, under the qubit labeling used by Qiskit, x would correspond to the outcome on qubit 2, y to the outcome on qubit 1, and z to the outcome on qubit 0.

Note: This representation of the bitstring puts the most significant bit (MSB) on the left, and the least significant bit (LSB) on the right. This is the standard ordering of binary bitstrings. We order the qubits in the same way (qubit representing the MSB has index 0), which is why Qiskit uses a non-standard tensor product order.

probability of obtaining outcome xyz is given by

Pr(xyz)=|⟨xyz|ψ⟩|SQR 2