Getting Started with Qiskit

2025-06-26

This tutorial provides a comprehensive introduction to getting started with quantum computing using Qiskit, IBM's open-source quantum computing framework. We demonstrate the complete workflow from installation and setup to implementing quantum circuits. The tutorial walks through creating a three-qubit Greenberger-Horne-Zeilinger (GHZ) entangled state using fundamental quantum gates including Hadamard and controlled-NOT operations. Using Qiskit Aer's simulation backends, we visualize quantum states, perform measurements, and analyze statistical outcomes with confidence intervals.

blog: https://tetraquark.vercel.app/posts/qiskitintro/?src=pdf

email: quarktetra@gmail.com

Introduction

Qiskit is a Python-based, open-source framework for quantum computing. It supports superconducting qubits and trapped ions as physical implementations. The qiskit library is available at GitHub **?**. This post is a detailed exploration of one of the examples **?** included the qiskit tutorials. All the credit for the code belongs to the qiskit team, and I made only small changes. If you would like to run the original code online, I pulled it into a binder page **?**.

Installation and Setup

Note: This setup was tested on macOS with Apple Silicon (M1/M2 chips) using conda environment.

Step 1: Install Qiskit and Qiskit Aer

In Qiskit 2.x, the simulator backend (Aer) has been moved to a separate package. You need to install both:

```
conda install -c conda-forge qiskit qiskit-aer -y
```

Alternatively, you can use pip, but conda is recommended for Apple Silicon Macs:

```
pip install qiskit qiskit-aer
```

Step 2: Configure Python Path for R/reticulate

If you're using R with reticulate (as in this document), you need to set the correct Python path:

```
# Set Python path to your conda installation
Sys.setenv(RETICULATE_PYTHON="/opt/anaconda3/bin/python")
```

To find your Python path, run in terminal:

```
which python3
# or
python3 -c "import sys; print(sys.executable)"
```

Step 3: Verify Installation

Test that everything works:

```
import qiskit
from qiskit_aer import Aer
print(f"Qiskit version: {qiskit.__version__}")
```

```
Qiskit version: 2.1.0
```

```
print(f"Available backends: {Aer.backends()}")
```

```
Available backends: [AerSimulator('aer_simulator'), AerSimulator('aer_simulator_statevector'
```

Important Changes in Qiskit 2.x

- **Aer import**: Use `from qiskit_aer import Aer` instead of `from qiskit import Aer`
- **Circuit execution**: Use `backend.run(circuit)` instead of `execute(circuit, backend)`
- **Circuit composition**: Use `circuit1.compose(circuit2)` instead of `circuit1 + circuit2`

Once you have the library properly installed, you can start playing with the code.

Quantum Gates

Any quantum algorithm will involve single qubit and multi qubit operations. This introduction code, for example, has a Hadamard-Walsh gate, which transforms $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. There will be other operations that act on two qubits at once, such as the controlled-not (CNOT) gate. We will start from a state $|\psi\rangle_f$ which is a tensor product of 3 independent states, initialized at $|0\rangle$, i.e.:

$$|\psi\rangle_i = |000\rangle. \tag{1}$$

After a set of gate operations, we will arrive at a three-qubit GHZ state, which is given by

$$|\psi\rangle_f = \frac{|000\rangle + |111\rangle}{\sqrt{2}}. \tag{2}$$

We can then simulate the circuit and make measurements on the outputs. The physical implementation of the qubits and the measulrement is beyond the scope of this post, however, I already have a post on superconducting qubits which might be a good read.

Let the code begin

We load the library and initiate a quantum system of 3 qubits.

```
import numpy as np
import matplotlib.pyplot as plt
from qiskit import QuantumCircuit
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit_aer import Aer


circ = QuantumCircuit(3) # Create a Quantum Circuit acting on a quantum register of three qu
```

The sequence of operations is as follows: