Quantum Computing with
the IBM Quantum Experience with the
Quantum Information Software Toolkit (QISKit)

Nick Bronn
Research Staff Member
IBM T.J. Watson Research Center
ACM Poughkeepsie Monthly Meeting, January 2018
Overview

Part 1: Quantum Computing
- What, why, how
- Quantum gates and circuits

Part 2: Superconducting Qubits
- Device properties
- Control and performance

Part 3: IBM Quantum Experience
- Website: GUI, user guides, community
- QISKit: API, SDK, Tutorials
Quantum computing: what, why, how
“Nature isn’t classical . . . if you want to make a simulation of nature, you’d better make it quantum mechanical, and by golly it’s a wonderful problem, because it doesn’t look so easy.”
– Richard Feynman, 1981
Computing with Quantum Mechanics: Features

Superposition: a system’s state can be any linear combination of classical states ...
...until it is measured, at which point it collapses to one of the classical states

Example: Schrödinger’s Cat

Entanglement: particles in superposition can develop correlations such that measuring just one affects them all

Example: EPR Paradox (Einstein: “spooky action at a distance”)
Computing with Quantum Mechanics: Drawbacks

Decoherence: a system is gradually measured by residual interaction with its environment, killing quantum behavior.

Consequence: quantum effects observed only in well-isolated systems (so not cats... yet)

Uncertainty principle: measuring one variable (e.g. position) disturbs its conjugate (e.g. momentum)

Consequence: complete knowledge of an arbitrary quantum state is impossible.

→ “No-Cloning Theorem”
What does a quantum bit look like?

**Classical bit**

**Physical systems:** capacitor charge, transistor state, magnetic polarization, presence or absence of a punched hole, etc.

**Logical states:** just 0 and 1

**Multi-bit effects:** none

**Quantum bit (“qubit”)**

**Physical systems:** electron spins, atomic states, *superconducting circuit states*

**Logical states:** $|0\rangle$, $|1\rangle$, *superpositions*

**Multi-qubit effects:** *entanglement*
Gate model quantum computing: the future

Today
~ 10 Qubits

Near future:
50-100 qubits too big to simulate!

Future:
Millions of qubits fully fault-tolerant

Fault-Tolerant QC
How powerful is a quantum computer: \textit{quantum volume}

Quantum Volume

- Number of \textbf{qubits} (more is better)
- \textbf{Errors} (fewer is better)
- \textbf{Connectivity} (more is better)
- \textbf{Gate set} (more is better)
Quantum computing: quantum operations and circuits
Single-qubit gates

- Gates are described by one or more rotations about an axis or set of axes
  - Pauli X, Y, Z gates:
    - Rotate $\pi$ radians about specified axis
    - X and Y gates equivalent to classical NOT
      - Transform $|0\rangle$ to $|1\rangle$ and vice versa
  - Clifford gates:
    - Permute states identified at right (includes Pauli gates)
  - Arbitrary gates:
    - Map any point on sphere to any other
    - Typically implemented with a small set of well-calibrated gates, e.g. **Clifford group** plus one additional gate

*Clifford group*: permutes the states $|0\rangle$, $|1\rangle$, $|+\rangle$, $|-\rangle$, $|\uparrow\rangle$, and $|\downarrow\rangle$, identified below

\[
\begin{align*}
|+\rangle &= |0\rangle + |1\rangle \sqrt{2} \\
|\downarrow\rangle &= |0\rangle - |1\rangle \sqrt{2} \\
|\uparrow\rangle &= |0\rangle + i |1\rangle \sqrt{2} \\
\end{align*}
\]
Key single-qubit gate: Hadamard \((H)\)

- **Hadamard** gate: rotate 180° about X+Z axis
  - Exchanges Z and X axes
  - Takes classical states to equal-weighted superposition states and vice versa
    - \(|0\rangle \rightarrow |+\rangle\)
    - \(|+\rangle \rightarrow |0\rangle\)
    - \(|1\rangle \rightarrow |-\rangle\)
    - \(|-\rangle \rightarrow |1\rangle\)
  - Used in almost every quantum algorithm

- Performs the **quantum Fourier transform** of a single qubit
  - Classical Fourier transform: exchange conjugate variables describing a *signal* (e.g. time domain \(\rightarrow\) frequency domain)
  - Quantum Fourier transform: exchange conjugate variables describing a *state*

Matrix representation of Hadamard acting on a single qubit:

\[
H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}
\]

\[
H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle
\]
Qubit measurements

- **Standard measurement in the computational basis:**
  - Collapses any superposition into one of the two classical states: $|0\rangle$ or $|1\rangle$

- **Measurement in other bases:**
  - Measurement itself is only sensitive to $|0\rangle$ vs $|1\rangle$
  - To measure in other bases, rotate first
  - Example: to distinguish $|+\rangle$ from $|-\rangle$, apply Hadamard before measuring
    - If state was $|+\rangle$, measure $|0\rangle$
    - If state was $|-\rangle$, measure $|1\rangle$
A simple “quantum score”

- Visual representation of a series of operations performed on a quantum register (a set of qubits grouped together)
- N-qubit quantum register: qubits q[0] – q[N-1]
- After measurement, results stored in classical register as c[0] – c[N-1]
- Example quantum score on 2-qubit register:
  - Initialize both qubits in $|0\rangle$
  - Apply Hadamard ($H$) to each qubit
  - Measure q[0] in the $|0\rangle$, $|1\rangle$ basis
  - Measure q[1] in the $|+\rangle$, $|−\rangle$ basis
- Results:
  - q[0] measurement gives either $|0\rangle$ or $|1\rangle$, each with 50% probability
  - q[1] measurement always gives $|0\rangle$
    - Infer that q[1] was in $|+\rangle$ prior to 2nd $H$
Multi-qubit operations

- Two-qubit operations:
  - Controlled not (CNOT):
    - Classical behavior: flip target iff control is 1
  - Controlled phase (CPhase)
    - Same idea but target qubit is flipped around the Z axis (instead of X)
    - Equivalent to CNOT up to single-qubit gates

<table>
<thead>
<tr>
<th>Initial State</th>
<th>Final State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Q</td>
<td>Control Q</td>
</tr>
<tr>
<td>Target Q</td>
<td>Target Q</td>
</tr>
<tr>
<td>$</td>
<td>\alpha + \beta</td>
</tr>
</tbody>
</table>

Entangled state!
Superconducting qubits: device properties
Superconducting qubit building blocks

Circuit element toolbox

| R | C | L | JJ |

Josephson Junction:
- Weak link between two superconductors
- Typically Al / AlOx / Al

Key features:
- non-linear inductance
- dissipationless operation

\[
\frac{dI}{dt} = \frac{1}{L} V(t) \\
L(\delta) = \frac{\Phi_0}{2\pi l_0 \cos(\delta)}
\]

L-C Oscillator: harmonic
→ can’t address individual transitions

\[
\omega_{23} = \omega_{12} \\
\omega_{12} = \omega_{01} \\
\omega_{01}
\]

JJ-C Oscillator: anharmonic
→ individual transitions addressable

\[
\omega_{23} \neq \omega_{01} \\
\omega_{12} \neq \omega_{01} \\
\omega_{01}
\]

Qubit
Qubit coupling via resonators: circuit QED (cQED)

- Qubit interacts with environment via a resonator
- Analogous to an atom in an optical cavity

\[ 2g = \text{vacuum Rabi freq.} \]
\[ \kappa = \text{cavity decay rate} \]
\[ \gamma = \text{“transverse” decay rate} \]

Jaynes-Cummings Hamiltonian

\[ \hat{H} = \hbar \omega_c (a^\dagger a + \frac{1}{2}) - \frac{\hbar \omega_0}{2} \hat{\sigma}_z - \hbar g (a^\dagger \sigma^- + \sigma^+ a) + H_\kappa + H_\gamma \]

- Quantized Field
- 2-level system
- Electric dipole interaction
- Dissipation

Qubit Readout in cQED

Create pulses → Readout pulses → Resonator / Qubit System → Amplify, digitize, identify as 0 or 1

Control pulses

Readout freq. near $\omega_r$; control freq. at $\omega_0$

Resonator frequency depends on qubit state → Infer qubit state from resonator response

$|0\rangle$ $|$ 1 $\rangle$

Amplitude

$\sqrt{I_m^2 + Q_m^2}$

2$\chi$

Phase (deg)

$\tan^{-1}(Q_m / I_m)$

$\theta = 2 \tan^{-1}(\chi / \kappa)$

For $2\chi = \kappa$, $\theta = 90^\circ$

$I = \text{in-phase}$

$Q = \text{out-of-phase}$

Gambetta et al., PRA 77, 012112 (2008)
Jeffrey et al., PRL 112, 190504 (2014)
Magesan et al., PRL 114, 200501 (2015)
IBM single-junction transmons

- Patterned superconducting metal (niobium + aluminum) on silicon
  - Qubit capacitance dominated by shunting capacitance $C_S$
- Resonant frequency $\sim 5 \text{ GHz} \rightarrow$ energy splitting $\sim 20 \ \mu\text{eV}$, or 240 mK
  $\rightarrow$ Cool in a dilution refrigerator ($\sim 10 \text{ mK}$) to reach ground state
- Interactions mediated by capacitively coupled co-planar waveguide resonators (circuit QED)
Anatomy of a multi-qubit device

**Qubits:**
- Single-junction transmon
- Frequency $\sim 5$ GHz
- Anharmonicity $\sim 0.3$ GHz

**Resonators:**
- Co-planar waveguide
- Frequency $\sim 6$ – $7$ GHz

**Roles:**
- Individual qubit readout
- Qubit coupling (“bus”)

**Ground plane**
- Periodic holes prevent stray magnetic field from hurting superconductor performance

Corcoles et al., Nat. Commun. 6, 6979 (2015)
IBM Quantum Experience
IBM Quantum Experience (IBMQX)

- Free cloud based quantum computing platform
  - 5-qubit quantum processor (real hardware)
  - 20-qubit quantum simulator
  - 16-qubit quantum processor (access through QISKit: www.qiskit.org)

IBM QX2: 5-qubit

Quantum Simulator

IBM QX3: 16-qubit
Performing Quantum Computing Experiments in the Cloud
Simon J. Devitt
Center for Emergent Matter Sciences, RIKEN, Wako-shi, Saitama 351-0198, Japan.

Quantum computing interest from both the experimental and theoretical aspects has recently gathered momentum. This has been accompanied by many companies to develop quantum computer hardware and software. In this contribution, we describe our recent experience with Quantum Cloud Services through IBM’s Quantum Experience and discuss the results of a theoretical study we performed on the problem of minimizing experimental error in a quantum circuit.

Experimental test of Mermin inequalities on a five-qubit quantum computer
Daniel Altman and José Ignacio Latorre
Dipartimento di Fisica Quantistica e Applicazioni, Università di Bologna, Diagonale 264, 40126 Bologna, Italy
Institute of Theoretical Physics, University of California, Berkeley, CA 94720, USA
(Rceived 25 May 2016; published 11 July 2016)

Mermin inequalities are a way to rule out local hidden variable theories for quantum systems. We have performed an experiment on a five-qubit quantum computer to test these inequalities.

New Journal of Physics
Entropic uncertainty and measurement reversibility
Martin Berta, Stephanie Wehner, and Mark M. Wilde
Center for Quantum Information and Signal Processing, Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA 93106, USA

We consider the problem of measuring a quantum system in a way that minimizes information leakage to an eavesdropper. We show that it is possible to measure a quantum system in such a way that the information leakage is zero.

Compressed quantum computing using the IBM Quantum Experience
M. Hohenstein, D. Almeida, J. I. Latorre, and B. Kraus
Institut de Física de Barcelona, Department of Theoretical Physics, University of Barcelona, Diagonal 647, 08028 Barcelona, Spain

We propose a new method for compressed quantum computing using the IBM Quantum Experience. Our method allows for the efficient execution of large-scale quantum circuits on a fixed number of qubits.

O Computador Quantico da IBM e o IBM Quantum Experience
Daniele S. Steiger and Matthias Troyer
Institute for Theoretical Physics, ETH Zürich, 8093 Zürich, Switzerland

We present an overview of the IBM Quantum Experience and describe the experimental setup used to perform the experiments.

Approximate Quantum Adders with Genetic Algorithms: An IBM Quantum Experience
Rui Li, Unal Alvarez-Rodriguez, Lucas Lamsa, and Enrique Solano
Department of Physics, University of the Basque Country UPV/EHU, Apartado 644, 48080 Bilbao, Spain

We present an algorithm for constructing approximate quantum adders using genetic algorithms. Our results show that the algorithm can be used to construct adders with a smaller number of qubits than previous methods.

Quantum state reconstruction made easy: a direct method for tomography
B. P. Lanyon, T. Timony, J. H. Siewert, and M. J. Everitt
Quantum Systems Engineering Research Group, Department of Physics, University of Oxford, Oxford, United Kingdom

We present a new method for reconstructing the state of a quantum system. Our method is based on the Direct Method for Tomography and is shown to be more efficient than previous methods.

Demonstration of entanglement assisted invariance on IBM’s Quantum Experience
Sebastian Deffner
Department of Physics, University of Maryland Baltimore County, Baltimore, MD 21250, USA

We demonstrate the principle of entanglement assisted invariance on IBM’s Quantum Experience. Our results show that the principle can be used to enhance the fidelity of quantum computations.

A quantum teleportation experiment for undergraduate students
S. Friletschenkova
Laboratoire d’Informatique de Paris-Diderot, Sorbonne Paris Cité, UMR 7089, CNRS, 75205 Paris, France

We report on a quantum teleportation experiment for undergraduate students. Our results show that the experiment can be successfully performed in a classroom setting.

Braiding Majoranas in a five qubit experiment
James R. Wootton
Department of Physics, University of Basel, Klingelbergstrasse 80, CH-4056 Basel, Switzerland
(Dated: September 19, 2017)

We report on a braiding experiment on a five-qubit device. Our results show that the experiment can be successfully performed on a quantum computer.
Real Quantum Processor: Device Details

• 5-qubit device
  – Single-junction transmons
  – $T_1 \sim T_2 \sim 50 – 100 \mu s$
  – 1Q gate fidelities $> 99\%$
  – 2Q gate fidelities $> 95\%$
  – Measurement fidelities $> 93\%$
  – Connectivity: 6 CNOTs available

• 16-qubit device (NEW!)
  – Access through QISKit API only
IBM QX: Web Interface

- [https://quantumexperience.ng.bluemix.net](https://quantumexperience.ng.bluemix.net)
- **Graphical composer**
  - Compose quantum circuits using drag and drop interface
  - Save circuits online or as QASM text, and import later
  - Run circuits on real hardware and simulator
IBM QX: Web Interface

- [https://quantumexperience.ng.bluemix.net](https://quantumexperience.ng.bluemix.net)
- Library
  - User guides for all levels (beginner, advanced, developer)
  - Run examples in composer
IBM QX: Web Interface

- [https://quantumexperience.ng.bluemix.net](https://quantumexperience.ng.bluemix.net)

- **Community forum**
  - Ask questions, discuss ideas
  - Receive answers from IBM staff and community members
  - Keep up to date with announcements and news
IBM QX: QISKit Interface

- [www.qiskit.org](http://www.qiskit.org)
- Open source project for quantum software development tools
IBM QX: QISKit Interface

- [www.qiskit.org](http://www.qiskit.org)

- GitHub: Python SDK
  - Advanced interface interacting with quantum hardware and simulators through python.
  - Write hybrid quantum-classical programs

---

**Qiskit**

Quantum Information Software Kit

Approximate Quantum Computing: From advantage to applications

Recordings now available!

**Latest version**

Inbox (489) - Nick.brom@gmail.com - Gmail: Qiskit for short is a software development kit (SDK) for working with OpenQASM and the IBM Q experience (QX).

**Learn**

Use QISKit to create quantum computing programs, compile them, and execute them on one of several backends (online, real quantum processors, and simulators).

**Run a quantum program**

```python
$ pip install qiskit
```

From qiskit import QuantumProgram
gp = QuantumProgram()

```
gr = gp.create_quantum_register("qr", 2)
cr = gp.create_classical_register("cr", 2)
sq = gp.create_circuit("qell", [qr], [cr])
sq.x(qr[0])
sq.x(qr[1])
sq.measure(qr[0], cr[0])
sq.measure(qr[1], cr[1])
result = gp.execute("qell")
print(result.get_counts("qell"))
```
IBM QX: QISKit Interface

- www.qiskit.org
- GitHub: Python SDK
  - Advanced interface interacting with quantum hardware and simulators through python.
  - Write hybrid quantum-classical programs
- GitHub: Tutorial Notebooks
  - Interactive Jupyter notebooks demonstrating a variety of topics

1. Introducing the tools

In this first topic, we break down the tools in the QISKit SDK, and introduce all the different parts to make this useful. Our list of introductory notebooks:

- Getting started with QISKit SDK - how to use QISKit SDK.
- Understanding the different backends - how to get information about the connected backends.
- Compiling and running a quantum program - how to rewrite circuits to different backends.
- Running a quantum program on IBM DSX - how to run a quantum notebook directly using IBM Data Science Experience (i.e. without installing any dependencies locally!)
- Loading and Saving a Quantum Program [coming soon].
- Visualizing a quantum state - illustrates the different tools we have for visualizing a quantum state.
- Quantum gates and linear algebra - list all basic gates and their definitions

2. Exploring quantum information concepts

The next set of notebooks shows how you can explore some simple concepts of quantum information science.
IBM QX: QISKit Interface

- [www.qiskit.org](http://www.qiskit.org)

- GitHub: Python SDK
  - Advanced interface interacting with quantum hardware and simulators through python.
  - Write hybrid quantum-classical programs

- GitHub: Tutorial Notebooks
  - Interactive Jupyter notebooks demonstrating a variety of topics
IBM QX: QISKit Interface

- **www.qiskit.org**
- **GitHub: Python SDK**
  - Advanced interface interacting with quantum hardware and simulators through python.
  - Write hybrid quantum-classical programs
- **GitHub: Tutorial Notebooks**
  - Interactive Jupyter notebooks demonstrating a variety of topics
- **Advanced documentation**
Using the Web Interface
**IBMQX: Getting started**

- Create account at [https://quantumexperience.ng.bluemix.net](https://quantumexperience.ng.bluemix.net)
- Create a new experiment: our example is 2-qubits

### New Experiment

<table>
<thead>
<tr>
<th>Quantum Registers</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>q</td>
<td>2</td>
</tr>
<tr>
<td>Number of Qubits</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**+ Add Quantum Register**

### Classical Registers

<table>
<thead>
<tr>
<th>Classical Registers</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>c</td>
<td>2</td>
</tr>
<tr>
<td>Number of bits</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**+ Add Classical Register**

**Set Topology**
IBMQX: Getting started

• Create account at https://quantumexperience.ng.bluemix.net

• Create a new experiment: our example is 2-qubits

Drag and drop gates onto the score
IBMQX: Getting started

• Create account at https://quantumexperience.ng.bluemix.net

• Create a new experiment: our example is 2-qubits

• Score is translated into OPENQASM (a Quantum Assembly Language) behind the scene
IBMQX: Getting started

• Create account at https://quantumexperience.ng.bluemix.net

• Create a new experiment: our example is 2-qubits

```
include "qelib1.inc";
qreg q[2];
creg c[2];
h q[0];
cx q[0],q[1];
t q[1];
cx q[0],q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

• Score is translated into OPENQASM (a Quantum Assembly Language) behind the scene
Basic Operation

• Universal gate set is available

Pauli gates
Clifford gates
Universal gate et

Barriers
Measurements
Basic Operation

- Universal gate set is available

Get additional information about gates by clicking here
Basic Operation

**U1**
The first physical gate of the Quantum Experience.
It is a one parameter single-qubit phase gate with zero duration.

**U3**
The third physical gate of the Quantum Experience.
This gate performs
\[[\cos(\theta/2),-\exp(1i\lambda)]\]

**id**
The identity gate performs an idle operation on the qubit for a time equal to one unit of time.

**X**
The Pauli $X$ gate is a $\pi$-rotation around the $X$ axis and has the property that $X \rightarrow -X$, $Z \rightarrow -Z$.
Also referred to as a bit-flip.

**Y**
The Pauli $Y$ gate is a $\pi$-rotation around the $Y$ axis and has the property that $X \rightarrow -X$, $Z \rightarrow -Z$.
This is both a bit-flip and a phase-flip, and satisfies $Y = XZ$.

**Z**
The Pauli $Z$ gate is a $\pi$-rotation around the $Z$ axis and has the property that $X \rightarrow -X$, $Z \rightarrow Z$.
Also referred to as a phase-flip.

**H**
The Hadamard gate has the property that it maps $X \rightarrow Z$, and $Z \rightarrow X$.
This gate is required to make superpositions.
Advanced Operations

- Advanced operations give access to arbitrary single qubit gates (u1, u2, u3)
- Advanced 2-qubit gate subroutines

This gate generates a maximally entangled Bell state from the initial state
Advanced Operations

- Advanced operations give access to arbitrary single qubit gates (u1, u2, u3)
- Advanced 2-qubit gate subroutines
Generating an entangled state

- Lets use the gate to make a maximally entangled state.
- We clear the score and drag the new subroutine onto score
Generating an entangled state

- Let's use the gate to make a maximally entangled state.
- We clear the score and drag the new subroutine onto score
- Next we add measurements
Generating an entangled state

• Now we choose the simulation or experiment parameters
• Choose number of shots

Click here to choose number of shots for simulation or experiment
Generating an entangled state

- Now we choose the simulation or experiment parameters
- Choose number of shots
Generating an entangled state

• Now we choose the simulation or experiment parameters
• Choose number of shots
• Click simulate to run simulation

Name your experiment

Your experiment will be saved before executing it. Please, enter a name for your experiment:

Experiment name

Bell Experiment

Simulate

Shots: 100
Seed: Random

Edit parameters

qreg q[2];
creg c[2];
gate entU q[0],q[1] {
h q[0];
cx q[0],q[1];
}
entU q[0],q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
Experiment Results

- After running we may view the experiment results

Quantum State: Computation Basis

Count data can be exported as CSV file

Download CSV
Experiment Results

- After running we may view the experiment results
- Results are saved to your account to view or run again later
Using the QISKit SDK
QISKit: Getting started

- Download qiskit-tutorial from [https://github.com/QISKit/qiskit-tutorial](https://github.com/QISKit/qiskit-tutorial)
- Install qiskit (optionally download SDK from [https://github.com/QISKit/qiskit-sdk-py](https://github.com/QISKit/qiskit-sdk-py))
- Navigate to qiskit-tutorial folder and launch Jupyter notebook

```
1. cjwood@christophers-MacBook-Pro:~/Documents/IBM-Git/qiskit-tutorial
   → qiskit-tutorial git:(master) x pip install qiskit; jupyter notebook
```

- Create a new Python 3 notebook and import qiskit

```python
# Import QISKit
import qiskit
from qiskit import QuantumProgram  # basic QISKit object

# Add IBMQX API token and URL. Needed for online access
API_TOKEN = "your_quantum_experience_api_token_here"
API_URL = 'https://quantumexperience.ng.bluemix.net/api'
```
Programming a Quantum Experiment

The most important part of QISKit is the `QuantumProgram` class.
- Roughly equivalent to the score on web interface
- Used to build and store quantum circuits
- Import or export QASM
- Interface with backends to run experiments (on real hardware or simulators)

Designing an experiment
1. Create a new `QuantumProgram`
2. Add 1 or more quantum registers
3. Add 1 or more classical registers

```python
In [2]:
# Initialize a new quantum program
qp = QuantumProgram()

# Add a 2-qubit quantum register "qr"
qr = qp.create_quantum_register("qr", 2)

# Add a 2-bit register "cr" to record results
cr = qp.create_classical_register("cr", 2)
```
Programming a Quantum Experiment

Adding a circuit to a QuantumProgram

• Next we create a new circuit to prepare a 2-qubit entangled state: $|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$

• We must first create an empty circuit with a name (label). We use “example”

• Use circuit methods to add gates to the circuit:

```python
In [3]:
# Create a new empty circuit which uses these registers
circ = qp.create_circuit('example', [qr], [cr])
circ.h(qr[0])  # add Hadamard gates to qubit-0
circ.cx(qr[0], qr[1])  # CNOT between qubit-0 and qubit-1
circ.measure(qr, cr)  # measure qubits
```

Available circuit operation methods:

• Single qubit gates (iden., x, y, z, h, s, sdg, t, tdg, u1, u2, u3)

• Two qubit gates (cx, cy, cz, cu1, cu2)

• Measurement, reset, and barrier (measure, reset, barrier)
Programming a Quantum Experiment

• The quantum program now contains a single circuit that we may view:

```
In [4]: qp.get_circuit_names()
...
Out[4]: dict_keys(['example'])
```

• We may also view the QASM for this circuit:

```
In [5]: qasm = qp.get_qasm('example')
print(qasm)

OPENQASM 2.0;
include "qelib1.inc";
qreg qr[2];
creg cr[2];
h qr[0];
cx qr[0],qr[1];
measure qr[0] -> cr[0];
measure qr[1] -> cr[1];
```
Programming a Quantum Experiment

Executing the circuit on a simulator

• We may view available backends for running a circuit:

```python
In [6]: qp.available_backends()
Out[6]: ['local_qasm_cpp_simulator', 'local_qasm_simulator', 'local_unitary_simulator']
```

• To use online backends we must set our API token and URL as follows:

```python
In [7]: qp.set_api(API_TOKEN, API_URL)
qp.available_backends()
Out[7]: ['ibmqx3', 'ibmqx2', 'ibmqx_qasm_simulator', 'local_qasm_cpp_simulator', 'local_qasm_simulator', 'local_unitary_simulator']
```
Programming a Quantum Experiment

Executing the circuit on a simulator

- We will run on the 'local_qasm_simulator' which is an offline Python simulator.
- This is done using the `execute` command and returns a dictionary containing results:

```python
In [7]: backend = 'local_qasm_simulator'
shots = 1024
results = qp.execute('example', backend='local_qasm_simulator', shots=1024)
data = results.get_data('example')
print(data)

{'counts': {'00': 509, '11': 515}}
```

- The results contain a list of counts.
- Counts can also be accessed directly by method: `results.get_counts('example')`
- **Note:** Different backends may return different types of results in the data dictionary
- **Note:** A list of many circuits can be submitted at once by the execute command
Simulator Features

We claimed that we prepared an entangled state? How can we verify this?

- Using the simulator in QISKit we may cheat and look directly at the state:
- To do this create new circuit to prepare the state *without measurement*:

```python
In [8]: # Create a new empty circuit which uses these registers
circ = qp.create_circuit('bell', [qr], [cr])
circ.h(qr[0]) # add Hadamard gates to qubit-0
circ.cx(qr[0], qr[1]) # CNOT between qubit-0 and qubit-1
```

- **Execute**: using shots = 1 to obtain the quantum state vector

```python
In [9]: # Execute on simulator for 1 shot
backend = 'local_qasm_simulator'
shots = 1
results = qp.execute('bell', backend='local_qasm_simulator', shots=shots)
data = results.get_data('bell')
print(data)

{'quantum_state': array([ 0.70710678+0.j, 0.00000000+0.j, 0.00000000+0.j, 0.70710678+0.j]), 'classical_state': 0, 'counts': {'00': 1}}
```
Plotting States

Plotting a state using the Visualization module:

- The `qiskit.tools.visualization` model contains several methods of visualizing quantum states:

```python
In [10]:
    # Import QISKit visualization library
    from qiskit.tools.visualization import plot_state
    from qiskit.tools.qi.qi import outer

    # Plot the density matrix of the state
    rho = outer(data['quantum state'])  # convert to density matrix
    plot_state(rho, method='city')
```
Combining Circuits

How would we verify the state is entangled on a real experiment?

• We need to measure the state in different bases.
• Create a new measurement circuit

```
In [11]: measZZ = qp.create_circuit('measZZ', [qr], [cr])
measZZ.measure(qr, cr)
print(qp.get_gasm('measZZ'))

OPENQASM 2.0;
include "qelib1.inc";
qreg qr[2];
creg cr[2];
measure qr[0] -> cr[0];
measure qr[1] -> cr[1];
```
Combining Circuits

How would we verify the state is entangled on a real experiment?
• We need to measure the state in different bases.
• Create a new measurement circuit
• The measurement circuit can be appended to another circuit using the `+` operator
• This new circuit can be added to the quantum program using the `add_circuit` method

```
In [12]: qp.add_circuit('example_mZZ', circ + measZZ)
print(qp.get_qasm('example_mZZ'))

OPENQASM 2.0;
include "qelib1.inc";
qreg qr[2];
creg cr[2];
h qr[0];
cx qr[0],qr[1];
measure qr[0] -> cr[0];
measure qr[1] -> cr[1];
```
Combining Circuits

How would we verify the state is entangled on a real experiment?

• We need to measure the state in different bases.

• **We can repeat this for additional measurement circuits in different bases**

```python
In [13]:
# Create circuit to measure both qubits in X basis
measXX = qp.create_circuit('measXX', [qr], [cr])
measXX.h(qr)
measXX.measure(qr, cr)

# Create circuit to measure both qubits in Y basis
measYY = qp.create_circuit('measYY', [qr], [cr])
measYY.h(qr)
measYY.s(qr)
measYY.measure(qr, cr)

# Add circuits to QuantumProgram
qp.add_circuit('example_mXX', circ + measXX)
qp.add_circuit('example_mYY', circ + measYY)
print(qp.get_circuit_names())
dict_keys(['example', 'bell', 'measZZ', 'example_mZZ', 'measXX', 'measYY', 'example_mXX', 'example_mYY'])
```
Combining Circuits

How would we verify the state is entangled on a real experiment?

• We need to measure the state in different bases.

• **Run these circuits on a backend and get the counts:**

```python
In [14]:
backend = 'local_qasm_simulator'
shots = 1024
meas_circs = ['example_mZZ', 'example_mXX', 'example_mYY']
meas_res = qp.execute(meas_circs, backend=backend, shots=shots)

for c in meas_circs:
    print('Measured counts:', c)
    print(meas_res.get_counts(c))

Measured counts: example_mZZ
{'11': 517, '00': 507}

Measured counts: example_mXX
{'11': 520, '00': 504}

Measured counts: example_mYY
{'00': 498, '11': 526}
```
Explore QISKit

• What next?
• Explore the QISKit tutorial Jupyter notebooks. A good start are the ones in Section 2:

2. Exploring quantum information concepts

The next set of notebooks shows how you can explore some simple concepts of quantum information science.

• Superposition and Entanglement - how to make simple quantum states on one and two qubits, and demonstrates concepts such as quantum superpositions and entanglement.
• Single-qubit States: Amplitude and Phase - discusses more complicated single-qubit states.
• Single-qubit Quantum Random Access Coding - how superpositions of one-qubit quantum states can be used to encode two and three bits into one qubit, and how measurements can be used to decode any one bit with a success probability of more than half.
• Two-qubit Quantum Random Access Coding - how superposition and entanglement can be used to encode seven bits of information into two qubits, such that any one of seven bits can be recovered probabilistically.
• Entanglement Revisited - the CHSH inequality, and extensions for three qubits (Mermin).
• Quantum Teleportation - introduces quantum teleportation.
• Quantum Superdense Coding - introduces the concept of superdense coding.
• Quantum Fourier Transform - introduces the quantum Fourier transform.
• Vaidman Detection Test - demonstrates interaction free measurement through the Vaidman bomb detection test.