CSCI 4140
Fun With Qubits

Mark Green

Faculty of Science

Ontario Tech

Introduction

- In this lecture we will explore single qubits and the gates that can be applied to them
- This will be done in terms of Qiskit, so this is also an opportunity to learn the basics of Qiskit
- We will use Qiskit inside Jupyter Notebook
- This isn’t necessary, Qiskit can be used on its own in a standard Python program
- Jupyter Notebook just makes it much easier to try out different circuits

Qiskit

- Qiskit is a library that can be used with Python
- We use Python code to create quantum circuits, execute or simulate them and then analyze the results
- An instance of Python is running in the background behind our notebook
- Anything we compute in one cell can be used in subsequent cells
- Also when we load a notebook all the cells are executed

Qiskit

- The first cell in my notebooks contains all the import statements for the parts of Qiskit that I need

```
In [8]: from qiskit import QuantumCircuit, execute, Aer
        from qiskit.visualization import plot_histogram, plot_bloch_multivector
        from math import sqrt, pi
```

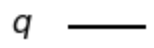
- If I find that I need additional parts of the library, I just add them to this cell and re-run it

Simple Circuits

- The simplest circuit contains a single qubit that does nothing
- We create a circuit by calling QuantumCircuit, the first parameter is the number of qubits, a second parameter is the number of classical bits

In [2]:
```
qc=QuantumCircuit(1)
qc.draw('mpl')
```

Out[2]:



- The draw procedure draws the circuit, a quick check is always a good idea

Simple Circuits

- In order to get any results we need to simulate the circuit
- Qiskit has a number of different simulators that do different things for us, we will use the state vector simulator for this example
- The simulator works in terms of the vectors and matrices that we’ve discussed in the main lecture
- Qiskit calls all its simulators, and the real quantum computers backends, so we need to get ourselves a backend and run a simulation

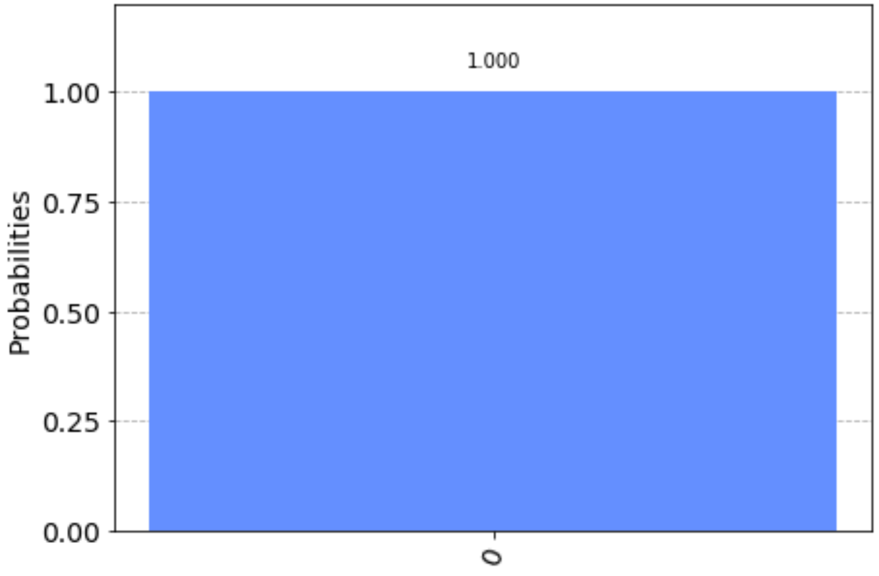
Simple Circuits

- The Aer.get_backend() procedure retrieves a backend, the parameter to this procedure is a text string, the name of the backend
- The execute procedure can then be used to execute the circuit, the first parameter is the circuit and the second parameter is the backend
- There are other parameters, mostly keyword, which we won't need
- The execute procedure returns a job object, on a real quantum computer we can use this object to determine when our job has been run
- The results() method on this object returns the results of our computation

Simple Circuits

```
qc=QuantumCircuit(1)
#qc.draw('mpl')
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
print(result.get_statevector())
plot_histogram(result.get_counts())
```

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



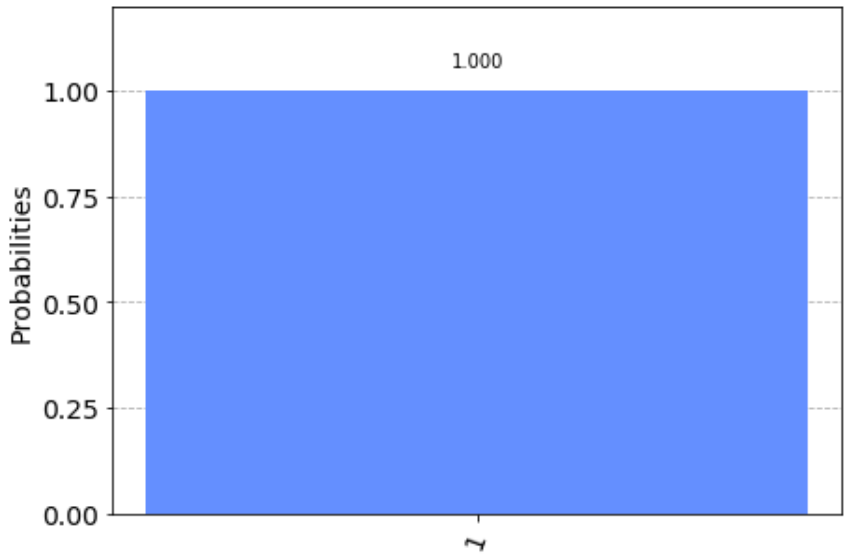
Simple Circuits

- One of the results we can retrieve is the state vector at the end of the simulation, which we can print
- Another result is the count of the number of times a particular result was seen
- In this case we know that the only possible result is |0>, which is what we get
- We can add an X gate to our circuit, so we will now get a |1>

Simple Circuits

```
qc=QuantumCircuit(1)
qc.x(0)
#qc.draw('mpl')
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
print(result.get_statevector())
plot_histogram(result.get_counts())
```

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



Simple Circuits

- To add the X gate we use the following:

  ```
  qc.x(0)
  ```

- This appends to X gate to the circuit operating on qubit 0
- Again we get the expected result, just the |1> value
- To get something more interesting we use the Hadamard gate, we replace the above statement by:

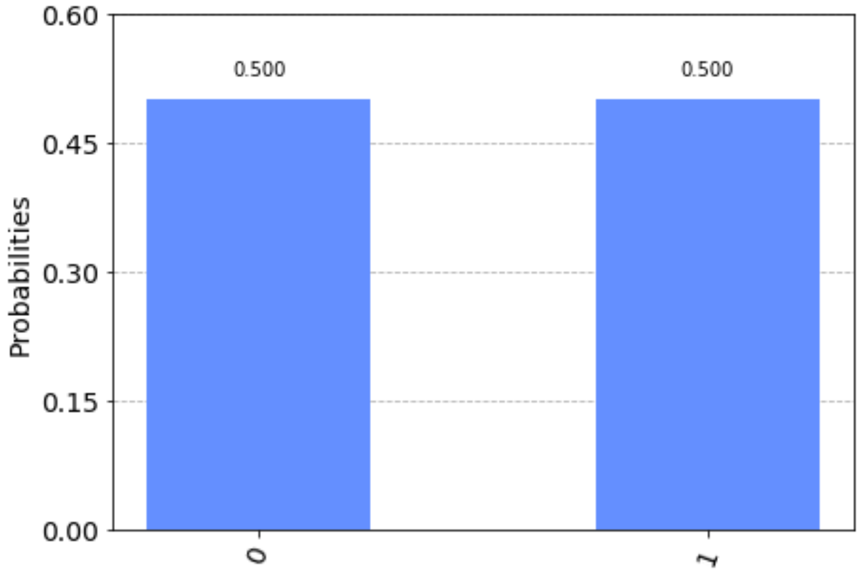
  ```
  qc.h(0)
  ```

- We now get an equal amount of |0> and |1>

Simple Circuits

```
qc=QuantumCircuit(1)
qc.h(0)
#qc.draw('mpl')
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
print(result.get_statevector())
plot_histogram(result.get_counts())
```

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

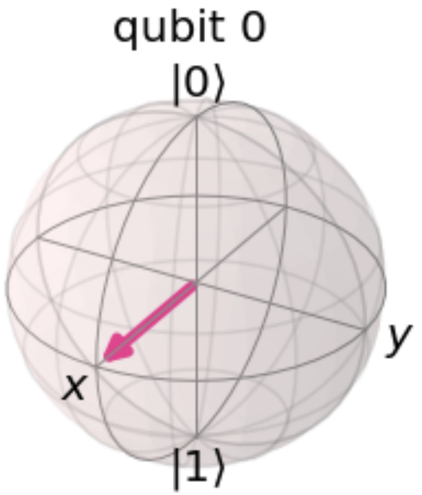


Simple Circuits

- Another way of viewing this result is with the Bloch sphere
- Qiskit has several Bloch sphere plotting functions, we will use the plot_bloch_multivector()
- This procedure takes a state vector and plots it on the Bloch sphere
- Note: a Jupyter Notebook cell can only have one visualization output
- If you try to have multiple ones only the last one will be shown, and it has to be the last statement in the cell
- This is not a bug, it’s a feature!

Simple Circuits

```
qc=QuantumCircuit(1)
qc.h(0)
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
plot_bloch_multivector(result.get_statevector())
```



Pauli Gates

- X, Y and Z are the Pauli gates given by the following matrices:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

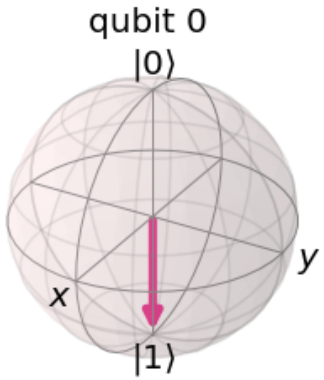
$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Pauli Gates

- We've already used the x() gate as you can guess there are also y() and z() gates

```
qc=QuantumCircuit(1)
qc.y(0)
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
plot_bloch_multivector(result.get_statevector())
```



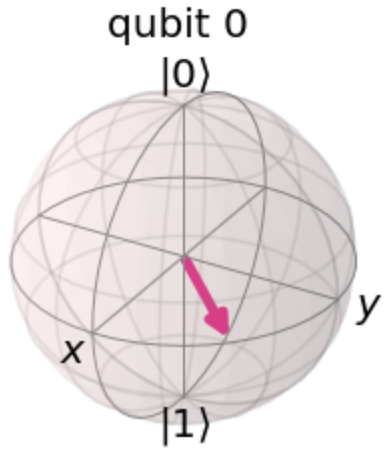
- This rotates a qubit around the y axis, the same goes for the z() gate

Rotations

- Basically the only thing we can do with a single qubit is rotations, so let’s examine some more rotation gates
- The rz() gate rotates about the Z axis by an arbitrary angle, the first parameter
- Recall that |0> and |1> lie along the Z axis, so if apply this gate to them we will see no difference
- We can use the h() gate to rotate |0> onto the positive x axis, so we can try this gate there

Rotations

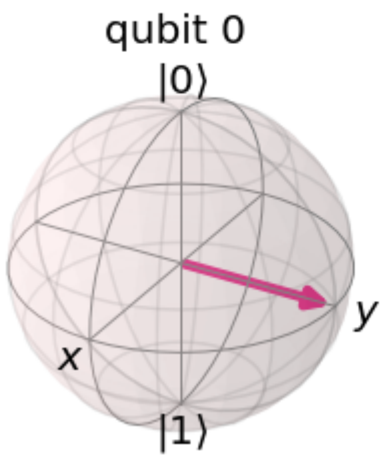
```
qc=QuantumCircuit(1)
qc.h(0)
qc.rz(pi/4,0)
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
plot_bloch_multivector(result.get_statevector())
```



Rotations

- You can try difference angles and see what you get
- Recall the S gate does a $\pi/2$ rotation about the Z axis

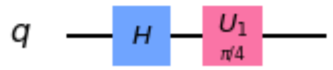
```
qc=QuantumCircuit(1)
qc.h(0)
qc.s(0)
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
plot_bloch_multivector(result.get_statevector())
```



Rotations

- There is also a t() gate that performs a π/4 rotation about z, give it a try
- We also had the U gates, with the U3 gates being universal, the following show what they look like in a circuit:

```
qc=QuantumCircuit(1)
qc.h(0)
qc.u1(pi/4,0)
qc.draw('mpl')
```



Summary

- Examined single qubits and the gates that can be applied to them
- Also started to work with Qiskit
  - How to create circuits
  - How to simulation circuits
  - How to visualize the results