CSCI 4140 Multiple Qubits

Mark Green

Faculty of Science

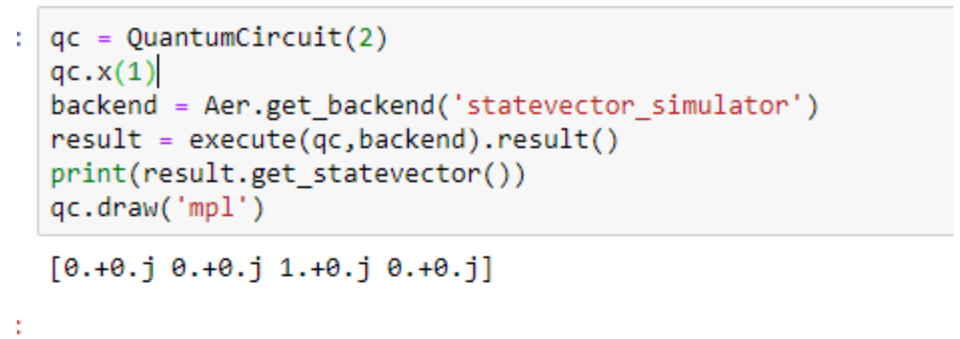
Ontario Tech

Introduction

- Things get interesting when we have more than one qubit
- We will examine 2 qubits in this lecture, everything generalizes nicely to an arbitrary number of qubits
- Start by creating a circuit with two qubits, the first in state |0> and the second in state |1>
- Simulate the circuit and print the resulting state vector

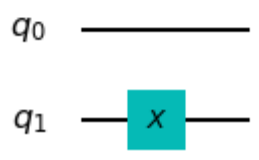
Introduction

- Note that we now have four values in our state vector
- Notice that there is only one non-zero entry in the state vector



```
qc = QuantumCircuit(2)
qc.x(1)
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
print(result.get_statevector())
qc.draw('mpl')
```

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



Introduction

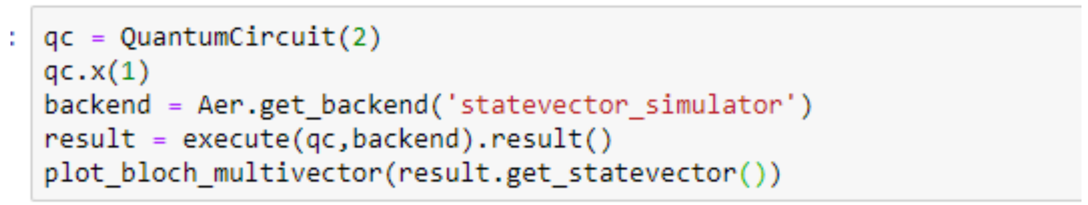
- The state vector that we got was (0, 0, 1, 0)^T, there are two qubits, but only one non-zero entry
- What’s going on here?
- The state vector for a 2 qubit system is given by:

$$|q\rangle = q_{00}|00\rangle + q_{01}|01\rangle + q_{10}|10\rangle + q_{11}|11\rangle$$

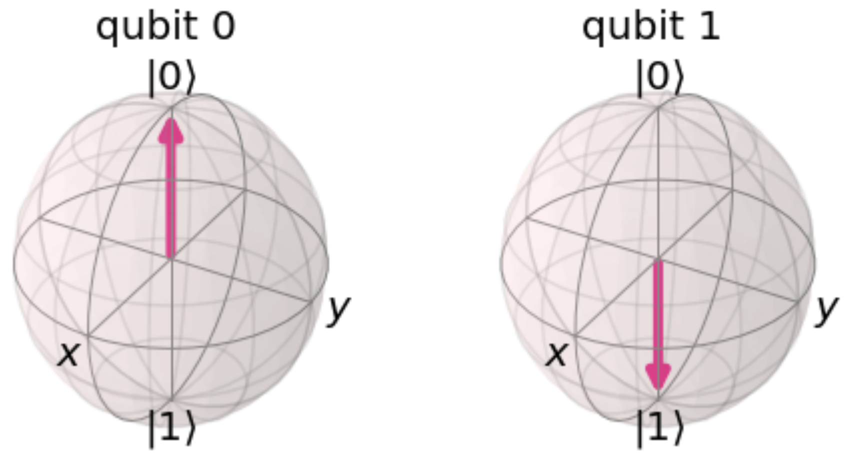
- It’s the |10> entry that has a non-zero entry, that’s because our first qubit has value 0 and our second qubit has value 1, it is one of the 4 possible combinations

Introduction

- We can also do a Bloch sphere of the results
- Note that in this case we get two Bloch spheres, one for each qubit



```
qc = QuantumCircuit(2)
qc.x(1)
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
plot_bloch_multivector(result.get_statevector())
```



Gates

- In this example we applied no gates to the first qubit, or the I gate
- We are applying the X gate to the second qubit
- Both of these gates are 2x2 matrices, but the state vector is of length 4, we can’t do the matrix multiplication
- We use the tensor product, in the case of |ba>, with M_b applied to |b> and M_a applied to |a>, we have the following:

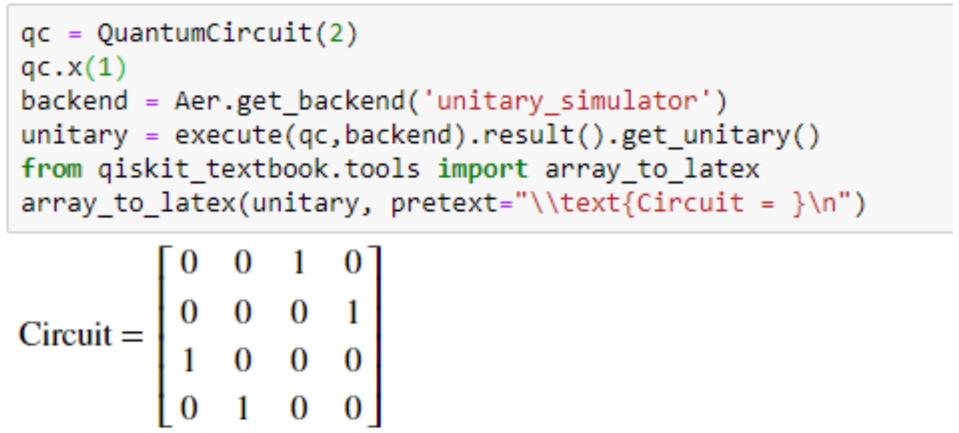
$$M = M_b \otimes M_a$$

- So in our case we want

$$M = X \otimes I$$

Gates

- There is a way of testing this in Qiskit, we use the unitary_simulator backend
- This backend produces the final matrix for the circuit

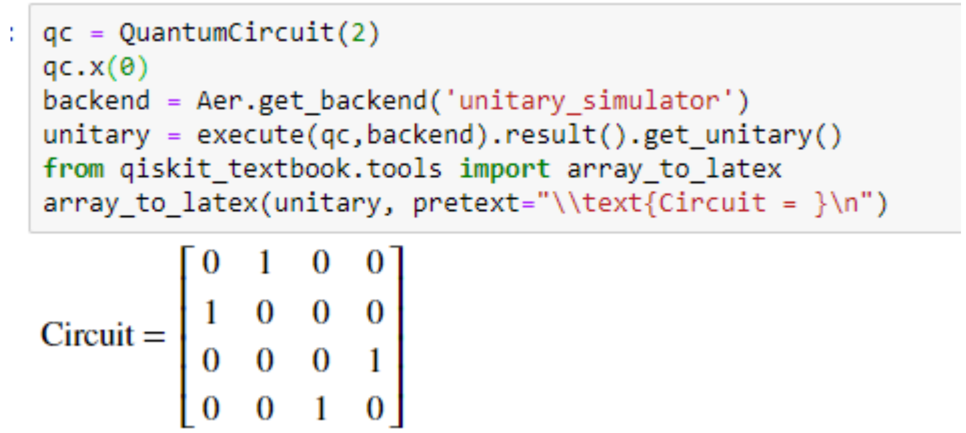


```
qc = QuantumCircuit(2)
qc.x(1)
backend = Aer.get_backend('unitary_simulator')
unitary = execute(qc,backend).result().get_unitary()
from qiskit_textbook.tools import array_to_latex
array_to_latex(unitary, pretext="\\text{Circuit = }\n")
```

$$\text{Circuit} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Gates

- If we put the X gate on the first qubit we get a slightly different result:



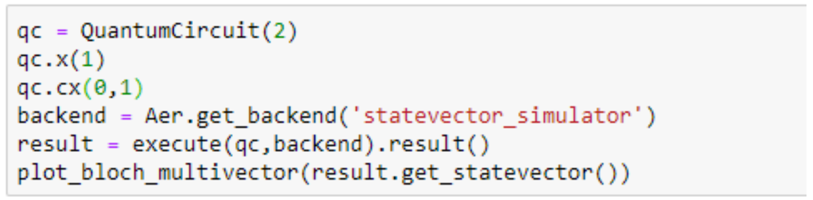
```
qc = QuantumCircuit(2)
qc.x(0)
backend = Aer.get_backend('unitary_simulator')
unitary = execute(qc,backend).result().get_unitary()
from qiskit_textbook.tools import array_to_latex
array_to_latex(unitary, pretext="\\text{Circuit = }\n")
```

$$\text{Circuit} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

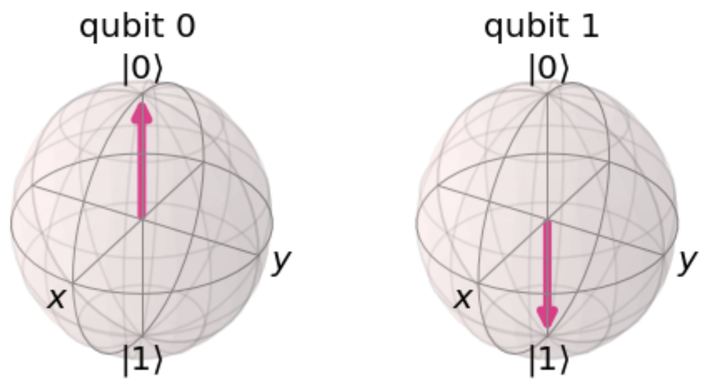
- Recall the that the tensor product isn’t communitive, so we get a different matrix

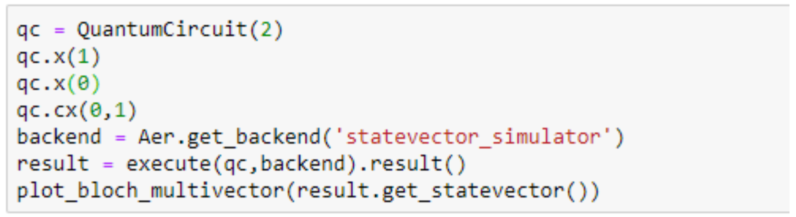
CNOT

- The controlled not or CNOT is the basic 2 qubit gate
- We noted that if the control qubit only has the values |0> and |1> this gate basically behaves as a not when the control is |1>

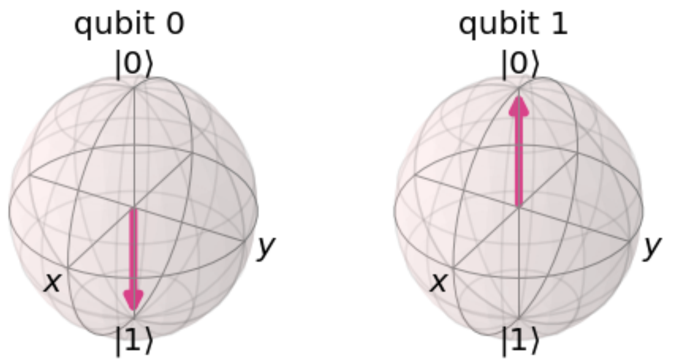


```
qc = QuantumCircuit(2)
qc.x(1)
qc.cx(0,1)
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
plot_bloch_multivector(result.get_statevector())
```



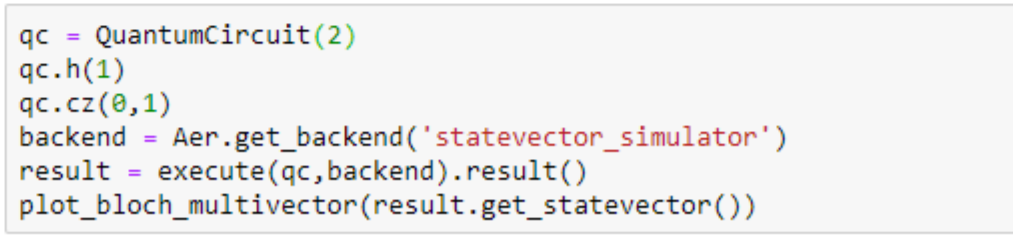


```
qc = QuantumCircuit(2)
qc.x(1)
qc.x(0)
qc.cx(0,1)
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
plot_bloch_multivector(result.get_statevector())
```

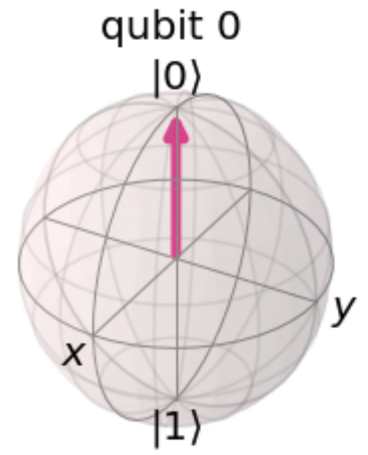


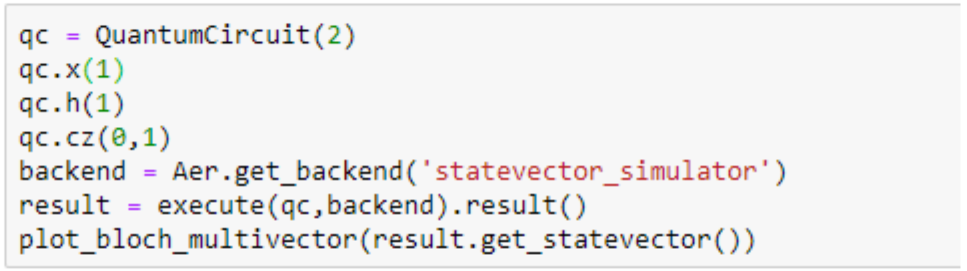
CNOT

- The CZ gate behaves the same way along the X axis

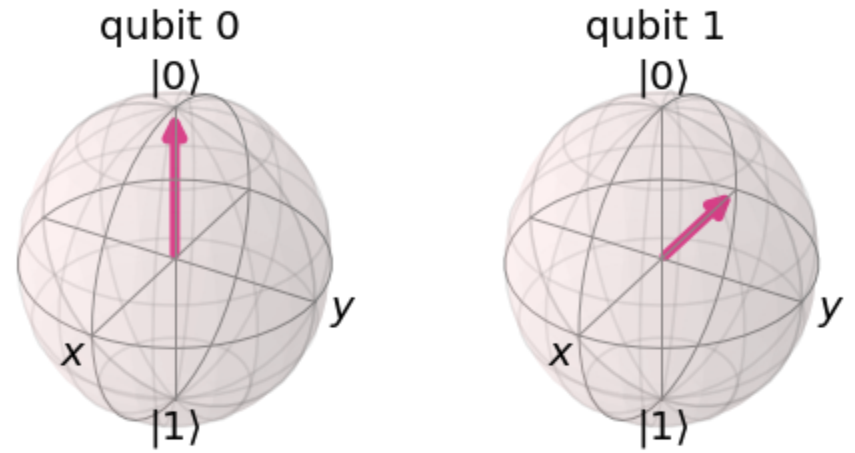


```
qc = QuantumCircuit(2)
qc.h(1)
qc.cz(0,1)
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
plot_bloch_multivector(result.get_statevector())
```



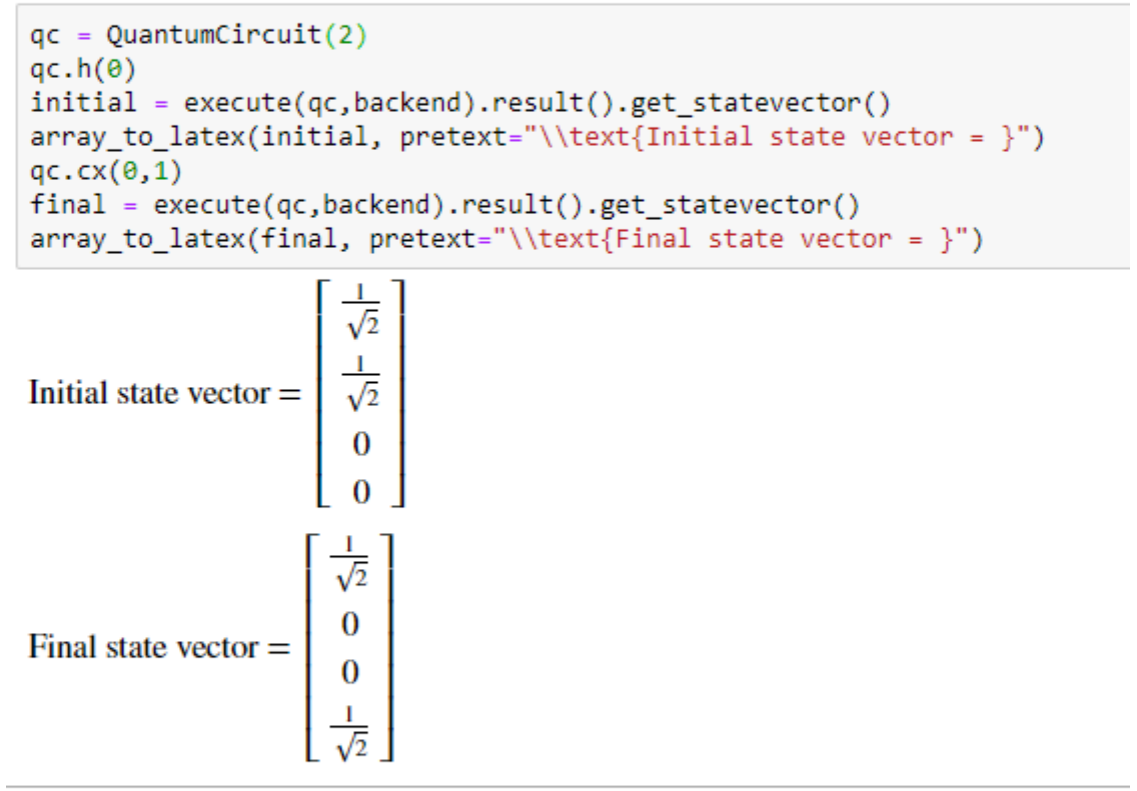


```
qc = QuantumCircuit(2)
qc.x(1)
qc.h(1)
qc.cz(0,1)
backend = Aer.get_backend('statevector_simulator')
result = execute(qc,backend).result()
plot_bloch_multivector(result.get_statevector())
```



CNOT

- Now lets return to the case of a superimposed control qubit, that is we apply the H gate to qubit zero



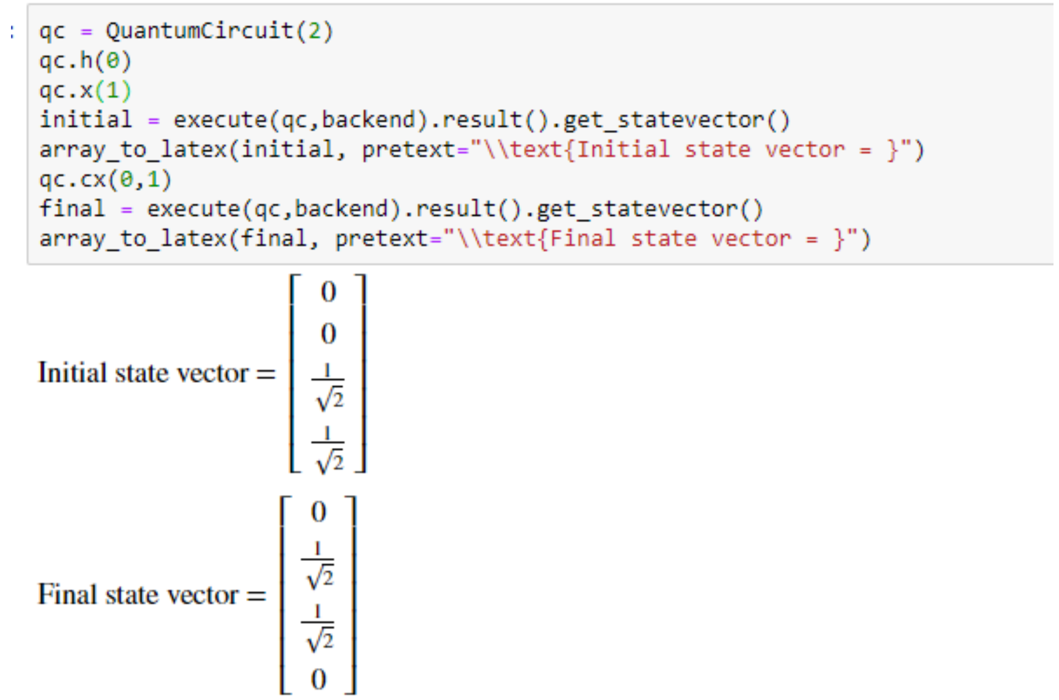
```
qc = QuantumCircuit(2)
qc.h(0)
initial = execute(qc,backend).result().get_statevector()
array_to_latex(initial, pretext="\\text{Initial state vector = }")
qc.cx(0,1)
final = execute(qc,backend).result().get_statevector()
array_to_latex(final, pretext="\\text{Final state vector = }")
```

$$\text{Initial state vector} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \\ 0 \end{bmatrix}$$

$$\text{Final state vector} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

CNOT

- This is one of the Bell states that we saw in the lecture
- We can get another one by applying an X gate to the second qubit



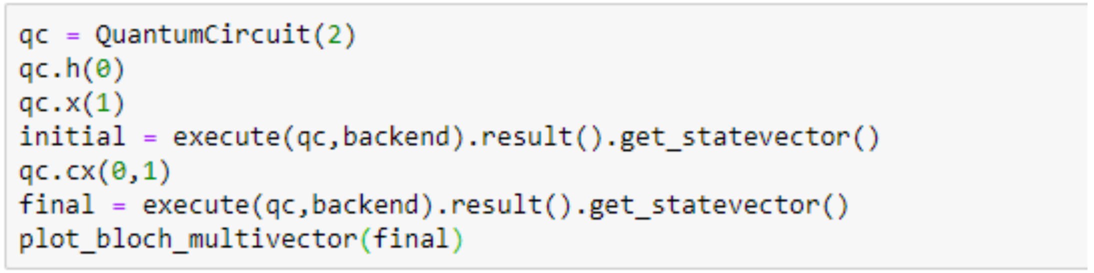
```
qc = QuantumCircuit(2)
qc.h(0)
qc.x(1)
initial = execute(qc,backend).result().get_statevector()
array_to_latex(initial, pretext="\\text{Initial state vector = }")
qc.cx(0,1)
final = execute(qc,backend).result().get_statevector()
array_to_latex(final, pretext="\\text{Final state vector = }")
```

$$\text{Initial state vector} = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

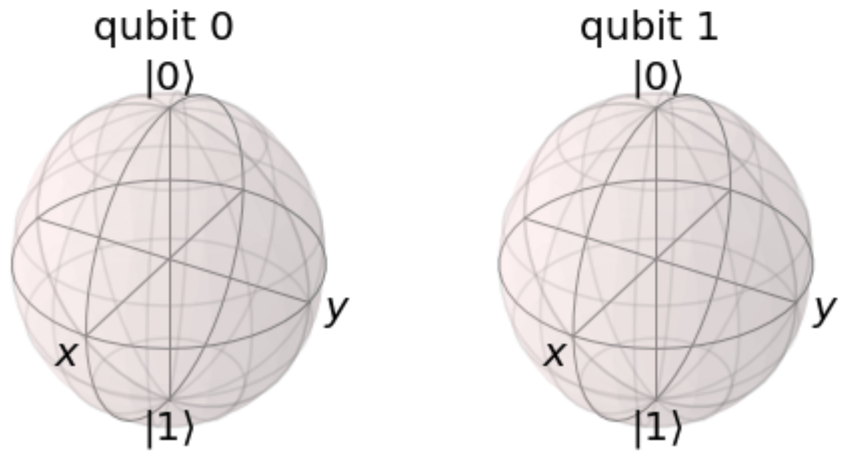
$$\text{Final state vector} = \begin{bmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}$$

CNOT

- Now lets see what happens when we try to plot the result on the Bloch sphere



```
qc = QuantumCircuit(2)
qc.h(0)
qc.x(1)
initial = execute(qc,backend).result().get_statevector()
qc.cx(0,1)
final = execute(qc,backend).result().get_statevector()
plot_bloch_multivector(final)
```

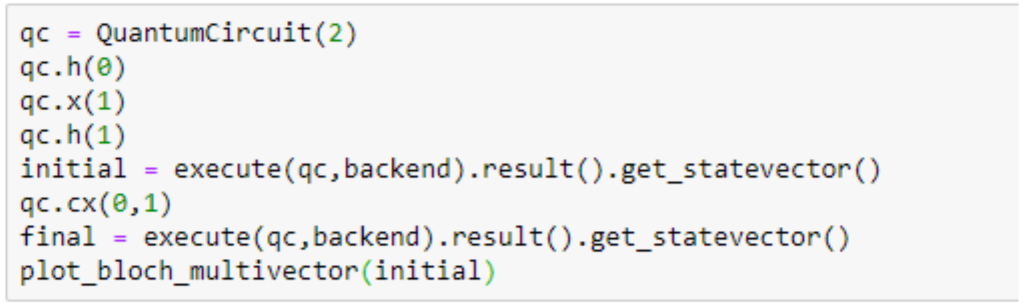


CNOT

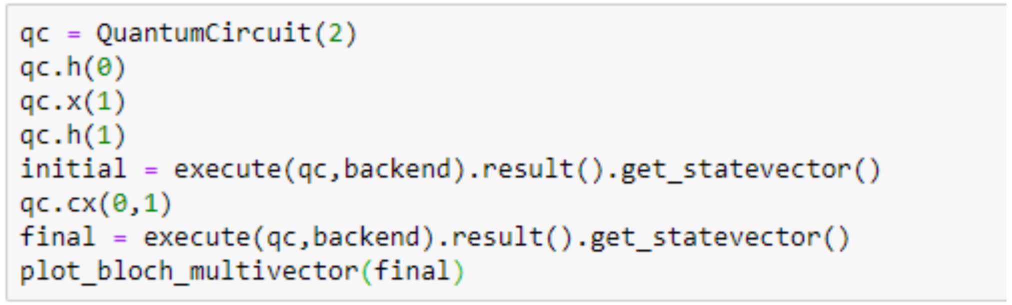
- What you see is not a bug!
- In a completely entangled state we know nothing about the states of the individual qubits, we only know the state of the complete system
- So, there is nothing for the Bloch sphere to show, it has no information on the individual qubits
- This is a bit surprising the first time that you get this result, but it is in fact correct

Phase Kickback

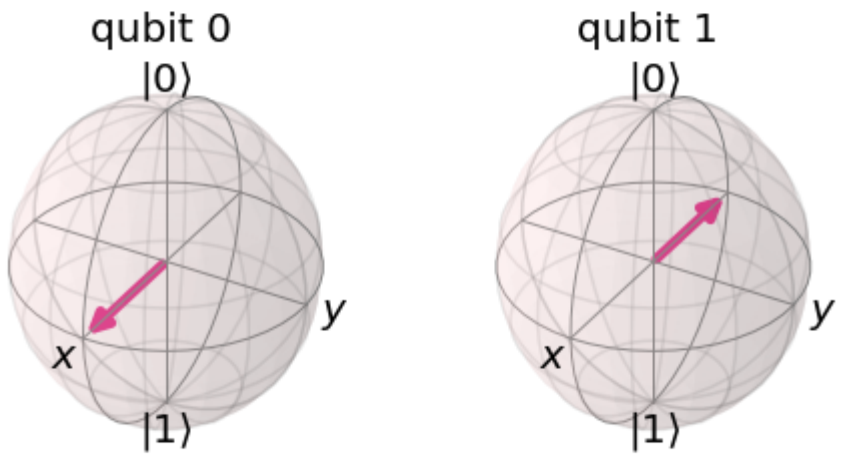
- Let’s repeat the phase kickback result that we had in the lecture

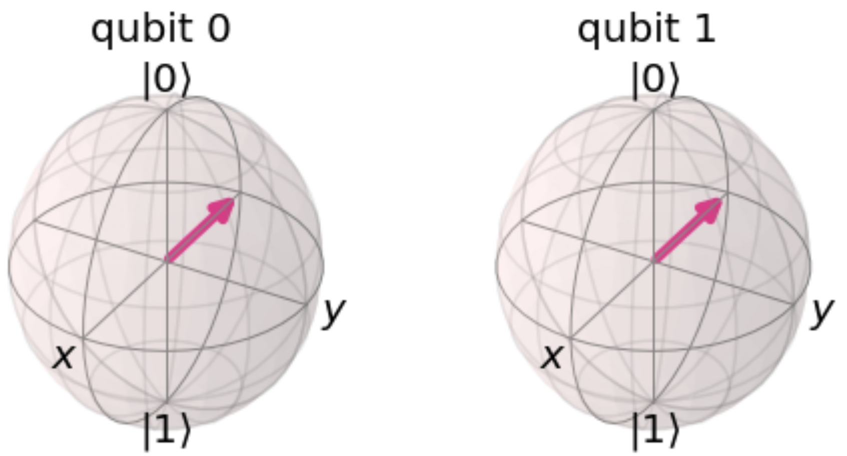


```
qc = QuantumCircuit(2)
qc.h(0)
qc.x(1)
qc.h(1)
initial = execute(qc,backend).result().get_statevector()
qc.cx(0,1)
final = execute(qc,backend).result().get_statevector()
plot_bloch_multivector(initial)
```



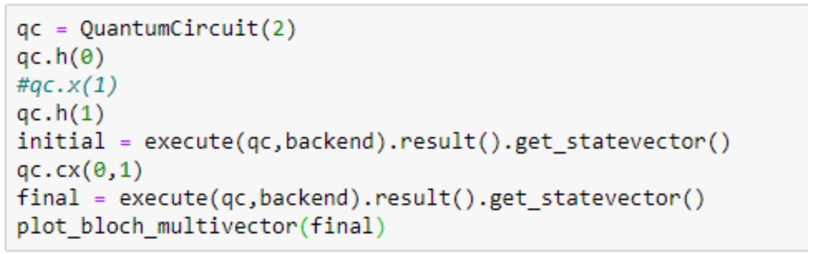
```
qc = QuantumCircuit(2)
qc.h(0)
qc.x(1)
qc.h(1)
initial = execute(qc,backend).result().get_statevector()
qc.cx(0,1)
final = execute(qc,backend).result().get_statevector()
plot_bloch_multivector(final)
```



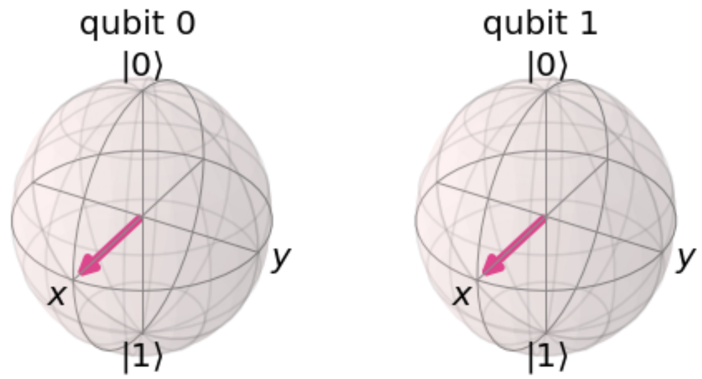


Phase Kickback

- Again we see that it’s the control qubit that changes
- If the second qubit is along the positive X axis, the first qubit doesn’t change so clearly the second qubit is the control one

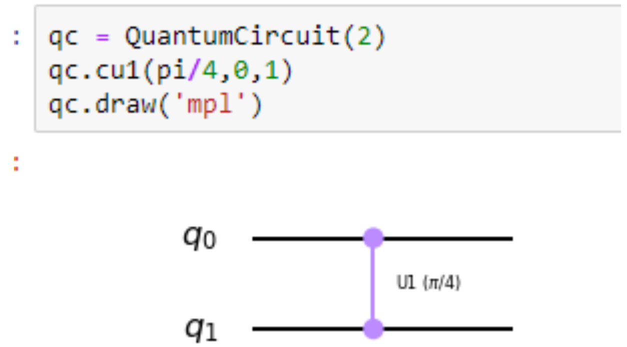


```
qc = QuantumCircuit(2)
qc.h(0)
#qc.x(1)
qc.h(1)
initial = execute(qc,backend).result().get_statevector()
qc.cx(0,1)
final = execute(qc,backend).result().get_statevector()
plot_bloch_multivector(final)
```



Phase Kickback

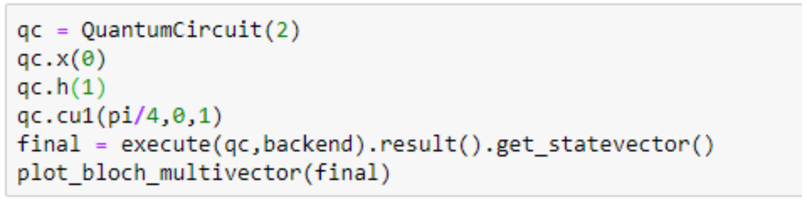
- Now lets examine the controlled T gate, recall that a T gate is just a U1 gate with the first parameter as π/4
- We can set up a controlled T gate in the following way:



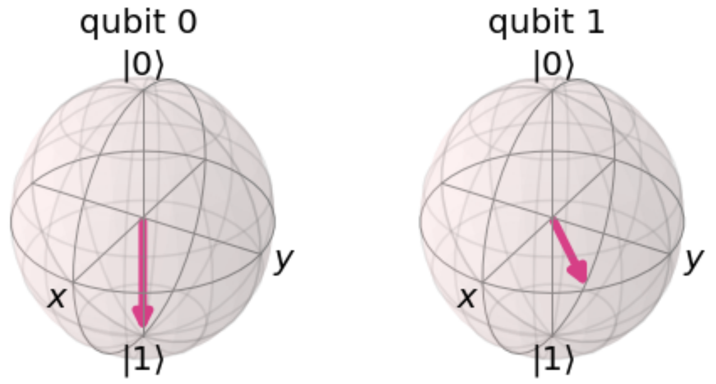
```
qc = QuantumCircuit(2)
qc.cu1(pi/4,0,1)
qc.draw('mpl')
```

Phase Kickback

- This circuit as it stands won’t do anything interesting, since it rotates about the Z axis and our qubits are along the Z axis
- In the following the second qubit is rotated on to the X axis and the gate behaves correctly

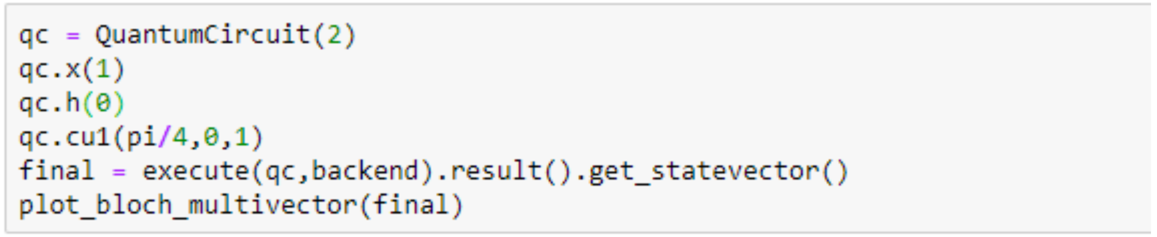


```
qc = QuantumCircuit(2)
qc.x(0)
qc.h(1)
qc.cu1(pi/4,0,1)
final = execute(qc,backend).result().get_statevector()
plot_bloch_multivector(final)
```

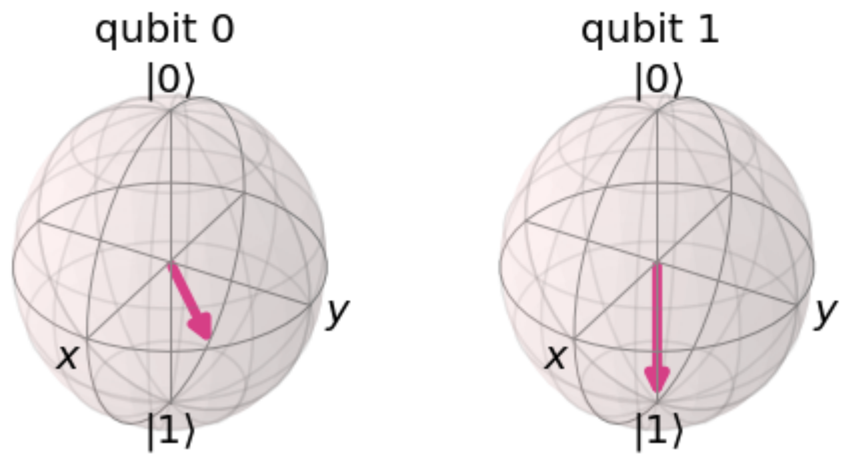


Phase Kickback

- Now lets flip the gates around so the H is on the first qubit, and we see that the first qubit is the one that rotates

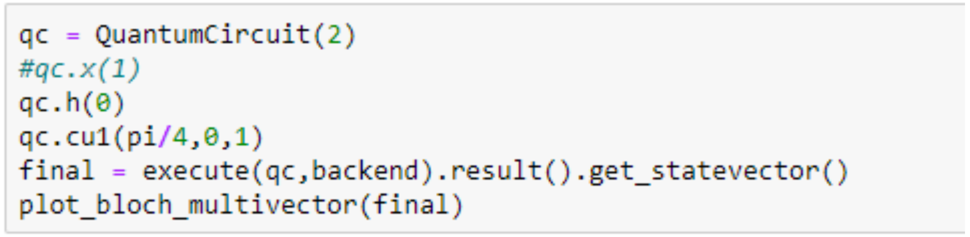


```
qc = QuantumCircuit(2)
qc.x(1)
qc.h(0)
qc.cu1(pi/4,0,1)
final = execute(qc,backend).result().get_statevector()
plot_bloch_multivector(final)
```

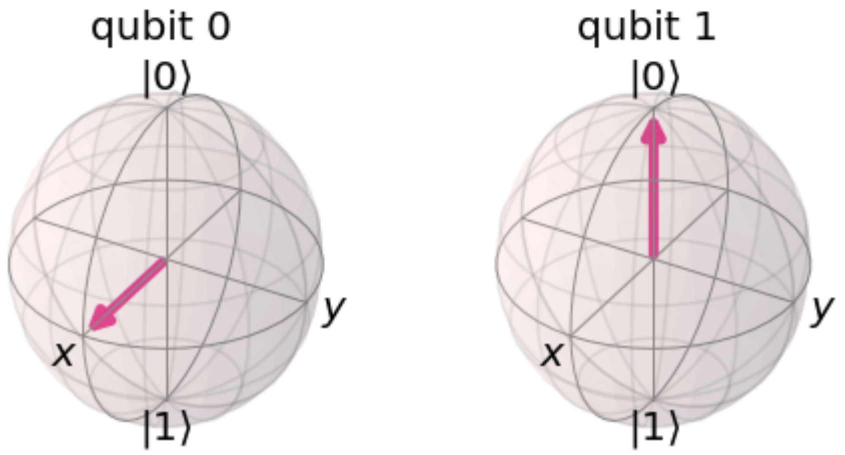


Phase Kickback

- If the X gate is taken off of the second qubit, the first qubit doesn't change, so clearly the second qubit is controlling the gate, and not the other way around



```
qc = QuantumCircuit(2)
#qc.x(1)
qc.h(0)
qc.cu1(pi/4,0,1)
final = execute(qc,backend).result().get_statevector()
plot_bloch_multivector(final)
```



Phase Kickback

- Phase kickback only occurs when the control qubit is in a superimposed state, it doesn’t happen if the control qubit is |0> or |1>
- This is the experimental side of phase kickback
- For the mathematical side see the phase kickback video lecture, make sure you view the eigenvalue and eigenvector video lecture first

Summary

- Examined how multiple qubits are represented in Qiskit, how the state vector is interpreted
- Examined how single qubit gates are applied to multiple qubits
- Examined controlled gates that operate on 2 qubits
- Examined phase kickback where the role of the control qubit is reversed