Qubits, Superposition, and Introduction to Qiskit

Julie Butler

Part 1: Defining Qubits in Python

A One Qubit System

- We will start our study of quantum computing by considering a one qubit system.
- This system can be represented as being in one of two states:

$$|\uparrow\rangle = |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |\downarrow\rangle = |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- We will go into physical implementations of quantum computers near the end of the course, but for now let's consider the qubit to be an electron and we are measuring and altering the spin in the z-direction.

Performing Calculations with One Qubit

- A one-qubit, two-state system has limited real world applications, but does allows us to get use to quantum computing with a simple system
- For a classical analog:
  - Bit: 0 Qubit: $|\uparrow\rangle$
  - Bit: 1 Qubit: $|\downarrow\rangle$

Defining the Two Qubits in Python

```
# Import Numpy for linear algebra
import numpy as np

# Define the two possible states of a single qubit
up = np.array([1,0])
down = np.array([0,1])

# Define a down qubit
qubit = down
print("Example Qubit:")
print(qubit)
```

See the Jupyter notebook associated with this lecture.

Part 2: Defining Quantum Gates in Python and Applying Them to Qubits

What are Quantum Gates?

- For a single qubit, a gate is a 2x2 matrix that manipulates a qubit
  - Like an operator BUT a gate does not collapse a superposition (there is no measurement being performed)
- Constraint: the matrices must be unitary ($U^\dagger U = \mathbf{I}$)
  - Quantum gates must be reversible

NOT Gate

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

- Function: Converts a qubit into the opposite qubit.
- $X|\uparrow\rangle = |\downarrow\rangle$ and $X|\downarrow\rangle = |\uparrow\rangle$
- In general: $X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$
  - Think of this in terms of superposition

Z Gate

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

- Function: Leaves $|\uparrow\rangle$ unchanged and converts $|\downarrow\rangle$ to $-|\downarrow\rangle$.

Hadamard Gate

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

- Question: What is the purpose of the -1?
- Function: Converts both $|\uparrow\rangle$ and $|\downarrow\rangle$ into states which are halfway between $|\uparrow\rangle$ and $|\downarrow\rangle$
  - $H|\uparrow\rangle = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$
  - $H|\downarrow\rangle = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$

Gate Operations in Python

```
# Define the NOT gate, the Z gate, and the Hadamard gate
X = np.array([[0,1],[1,0]])
Z = np.array([[1,0],[0,-1]])
H = (1/np.sqrt(2))*np.array([[1,1],[1,-1]])

# Check that each gate is unitary
print("X Unitary?")
print(X.T@X)
print("Z Unitary?")
print(Z.T@Z)
print("H Unitary?")
print(H.T@H)
```

Gate Operations in Python (Cont.)

```
# Apply the NOT gate to each single qubit
print("X and up")
print(X@up)
print("X and down")
print(X@down)

# Apply the Z gate to each single qubit
print("Z and up")
print(Z@up)
print("Z and down")
print(Z@down)
```

Gate Operations in Python (Cont.)

```
# Apply the Hadamard gate to each single qubit
print("H and up")
print(H@up)
print("H and down")
print(H@down)
```

See the Jupyter notebook associated with this lecture.

Part 3: Defining a Superposition of Qubits in Python

Superposition in Python

```
# Define a qubit using our previous computational basis
# Remember that in Python, j takes the place of the imaginary
# number i
qubit = 3*up + 4j*down
# Print the qubit and its norm
# Note that the qubit is not normalized
print("Superposition Qubit and Magnitude")
print(qubit)
print(np.linalg.norm(qubit))
```

Superposition in Python (Cont.)

```
# Normalize the qubit
mag = np.linalg.norm(qubit)
qubit = qubit/mag

# Print the normalized qubit and its new magnitude
print("Normalized Superposition Qubit and Magnitude")
print(qubit)
print(np.linalg.norm(qubit))
```

See the Jupyter notebook associated with this lecture.

Superposition in Python (Cont.)

```
# Find the probability that upon measurement, an up result is
# obtained
prob_up = np.dot(np.conjugate(up),qubit)*\
    np.conjugate(np.dot(np.conjugate(up),qubit))
# Find the probability that upon measurement, an down result i
# obtained
prob_down = np.dot(np.conjugate(down),qubit)*\
    np.conjugate(np.dot(np.conjugate(down),qubit))

# Print the probability of obtaining an up result, a down resu
# and ensure that the total probability is 1
print("Probability of obtaining up, probability of obtaining d
print(prob_up, prob_down, prob_up+prob_down)
```

A New Computational Basis

$$|+\rangle = \frac{1}{\sqrt{2}}(|\uparrow\rangle + |\downarrow\rangle) \quad |-\rangle = \frac{1}{\sqrt{2}}(|\uparrow\rangle - |\downarrow\rangle)$$

New Basis in Python

```
# Define the first state of our new computational basis and
# ensure that it is normalized
plus = (up + down)/np.sqrt(2)

print("+ qubit and magnitude")
print(plus)
print(np.linalg.norm(plus))
```

New Basis in Python (Cont.)

```
# Define the second state of our new computational basis and
# ensure that it is normalized
minus = (up - down)/np.sqrt(2)

print("- qubit and magnitude")
print(minus)
print(np.linalg.norm(minus))
```

Creating the New Basis on a Quantum Computer

- All qubits start out as $|\uparrow\rangle$, can be converted to $|\downarrow\rangle$ with a NOT gate
- Question: Can we create the $|+\rangle$ and $|-\rangle$ basis using our original basis ($|\uparrow\rangle$ and $|\downarrow\rangle$) and some number of quantum gates?

Hadamard Gate!

$$H|\uparrow\rangle = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} = \frac{1}{\sqrt{2}}(\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}) = \frac{1}{\sqrt{2}}(|\uparrow\rangle + |\downarrow\rangle)$$

$$H|\downarrow\rangle = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} = \frac{1}{\sqrt{2}}(\begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix}) = \frac{1}{\sqrt{2}}(|\uparrow\rangle - |\downarrow\rangle)$$

```
# Check that + and - can be built with up, down, and H
print(H@up == plus)
print(H@down == minus)
```

Part 4: Introduction to Qiskit and Drawing Quantum Circuits

Qiskit

- IBM developed Python library for simulating a quantum computer
- Can also use it to run a quantum circuit on a real IBM quantum computer

WARNING: IBM changed the syntax of Qiskit recently so many sources older than this year have code which will not run (or will not run correctly). Be careful which resources you are taking code from.

Drawing One Qubit Circuits

- The qubit is represented as a single horizontal line
- Gates are represented with boxes on that line (in the order they are applied) with a symbol representing the gate (X for NOT, Z for Z, H for Hadamard)
- An "M gate" represents measuring the state of the circuit.

Part 5: Introduction to One Qubit Quantum Circuits with Qiskit

Relevant Qiskit Imports

```
# Needed to set up the quantum circuit
from qiskit import QuantumCircuit, ClassicalRegister, QuantumR
# Needed to simulate running a quantum computer
from qiskit_aer import AerSimulator
# Neded to visualize the results of running a quantum computer
from qiskit.visualization import plot_histogram
```

- Note that only the first import is needed to create a quantum circuit, the second two are used to *run* the quantum circuit

Simulate One Qubit

```
# creates a Quantum register of 1 qubit
q = QuantumRegister(1)
# creates a classical register of 1 bit
c = ClassicalRegister(1)
# creates a quantum circuit that maps the result of a qubit
# to a classical bit
qc = QuantumCircuit(q, c)
# measure the current quantum circuit and draw a diagram
qc.measure(q, c)
print(qc.draw())
```

Simulate One Qubit (Cont.)

```
# Run the quantum circuit on a simulated quantum computer 1024
simulator = AerSimulator()
results = simulator.run(qc).result().get_counts()
# Create a histogram of the results
plot_histogram(results)
```

- All Qiskit qubits start in the $|\uparrow\rangle$ (0) state (notice there will be a slight bias to this state later)

Add a NOT Gate

```
# Create the same quantum circuit as before, but add a NOT gat
# measuring
q = QuantumRegister(1)
c = ClassicalRegister(1)
qc = QuantumCircuit(q, c)
qc.x(0)
qc.measure(q, c)
print(qc.draw())
simulator = AerSimulator()
results = simulator.run(qc).result().get_counts()
plot_histogram(results)
```

- Predicted result?

Add a Z Gate

```
# Replace the NOT gate with a Z gate
q = QuantumRegister(1)
c = ClassicalRegister(1)
qc = QuantumCircuit(q, c)
qc.z(0)
qc.measure(q, c)
print(qc.draw())
simulator = AerSimulator()
results = simulator.run(qc).result().get_counts()
plot_histogram(results)
```

- Predicted Result?

Add a Hadamard Gate

```
# Replace the Z gate with a Hadamard gate
q = QuantumRegister(1)
c = ClassicalRegister(1)
qc = QuantumCircuit(q, c)
qc.h(0)
qc.measure(q, c)
print(qc.draw())
simulator = AerSimulator()
results = simulator.run(qc).result().get_counts()
plot_histogram(results)
```

- Predicted Result?
- What state have we made?

What code could you use to create $|-\rangle$?

Why are we not running on a real quantum computer?

- Quantum Noise and Wait Times
- We will experiment with both simulate quantum noise and running on real quantum computers later in the semester once we have a better understanding!

Project: Create a Quantum Coin Flipper.

- It needs to return "Heads" or "Tails"
- Hint: to simulate the qubit once use

```
results = simulator.run(qc, shots=1).result().get_counts()
```

- Note that in the previous codes `results` is a dictionary of all outcomes.