The Quantum Fourier Transform (QFT) in Qiskit

A Single Qubit QFT

Introduction

The Quantum Fourier Transform (QFT) on a single qubit is straightforward, as it simply transforms the state of the qubit into the Fourier basis. For a 1-qubit system, the QFT is equivalent to the Hadamard gate, which creates a superposition by transforming $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$.

QFT Implementation for a 1-Qubit System in Qiskit

The following is a Qiskit implementation of a 1-qubit QFT, which for a single qubit is simply a Hadamard gate:

```
from qiskit import QuantumCircuit, Aer, execute
from qiskit.visualization import plot_bloch_multivector

# Create a 1-qubit quantum circuit
qc = QuantumCircuit(1)

# Apply the QFT, which for 1 qubit is simply a Hadamard gate
qc.h(0)

# Display the circuit
print("1-qubit QFT circuit:")
display(qc.draw('mpl'))

# Simulate and visualize the result on the Bloch sphere
simulator = Aer.get_backend('statevector_simulator')
result = execute(qc, simulator).result()
statevector = result.get_statevector()

# Plot the result on the Bloch sphere
print("Final state vector:", statevector)
plot_bloch_multivector(statevector)
```

Explanation of the Code

Quantum Circuit Setup

We create a `QuantumCircuit` with a single qubit.

Applying the QFT

For a 1-qubit system, the QFT is simply a Hadamard gate. The `qc.h(0)` line applies the Hadamard gate to qubit 0, transforming its state into an equal superposition of $|0\rangle$ and $|1\rangle$.

Simulation

We use the `statevector_simulator` to get the final state vector of the qubit after applying the QFT.

Visualization

We display the final state on the Bloch sphere using `plot_bloch_multivector(statevector)`. This shows the state in the Fourier basis, which for a 1-qubit system is the same as applying a Hadamard transform.

Explanation of Results

Final State

If the qubit starts in the $|0\rangle$ state, applying the Hadamard gate (or 1-qubit QFT) results in the state:

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

If it starts in $|1\rangle$, the final state will be:

$$\frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

For a single qubit, this transformation creates a superposition state in the Fourier basis, which is equivalent to the Hadamard gate.

A 2-Qubit QFT

Introduction

To implement the Quantum Fourier Transform (QFT) for a 2-qubit system in Qiskit, we need to apply a combination of Hadamard gates and controlled phase rotations. The 2-qubit QFT transforms the state into the Fourier basis, which involves creating superposition and phase relationships between both qubits.

QFT Circuit for 2 Qubits

For a 2-qubit system, the QFT circuit consists of:

- A Hadamard gate on the first qubit.
- A controlled phase rotation (CP) gate between the first and second qubits with a phase of $\pi/2$.
- A Hadamard gate on the second qubit.
- A swap gate to reverse the qubit order in the final output.

Here's the Qiskit code to implement this:

```
from qiskit import QuantumCircuit, Aer, execute
import numpy as np
from qiskit.visualization import plot_bloch_multivector

# Function to create a 2-qubit QFT circuit
def qft_2qubit():
    qc = QuantumCircuit(2)

    # Step 1: Apply Hadamard on the first qubit
    qc.h(0)

    # Step 2: Apply controlled-phase rotation with pi/2 on
        qubit 1 controlled by qubit 0
    qc.cp(np.pi / 2, 0, 1)

    # Step 3: Apply Hadamard on the second qubit
    qc.h(1)

    # Step 4: Swap the qubits to reverse their order
    qc.swap(0, 1)

    return qc

# Create and display the QFT circuit for 2 qubits
qc = qft_2qubit()
```

```
print("2-qubit QFT circuit:")
display(qc.draw('mpl'))

# Use the statevector simulator to observe the final state
simulator = Aer.get_backend('statevector_simulator')
qc_save_state = qc.copy()  # Copy circuit for observing
    statevector before measurement
result = execute(qc_save_state, simulator).result()
statevector = result.get_statevector()

# Display the final state vector and plot on Bloch sphere
print("Final state vector:", statevector)
plot_bloch_multivector(statevector)
```

Explanation of the Code

Hadamard on the First Qubit

The first Hadamard gate creates a superposition state on qubit 0, preparing it for the QFT transformation.

Controlled-Phase Rotation

The `qc.cp(np.pi / 2, 0, 1)` line applies a controlled-phase rotation with an angle of $\pi/2$ on qubit 1, controlled by qubit 0. This phase rotation establishes the Fourier basis relationships between the qubits.

Hadamard on the Second Qubit

A Hadamard gate is applied to qubit 1 to complete the superposition, creating the Fourier basis for the 2-qubit system.

Swap Gate

The `qc.swap(0, 1)` step swaps the two qubits to ensure the output order matches the Fourier-transformed basis, as the QFT generally outputs in reverse bit order.

Simulation

The circuit is simulated on the `statevector_simulator` to obtain the final state vector, which represents the state of the qubits after the QFT.

Explanation of the Result

The 2-qubit QFT transforms a 2-qubit system into the Fourier basis, establishing entanglement and phase relationships between the qubits. By examining the final state vector and visualizing the result on the Bloch sphere, we can observe how the QFT changes the quantum state based on its initial input.

A 3-Qubit QFT

Introduction

For a 3-qubit Quantum Fourier Transform (QFT), we apply Hadamard gates and controlled phase rotations among the qubits. The 3-qubit QFT involves more steps than the 2-qubit version, as each qubit interacts with the others through controlled-phase rotations.

Steps for 3-Qubit QFT

For a 3-qubit system, the QFT consists of the following:

1. A Hadamard gate on the first qubit, followed by controlled phase rotations with angles $\pi/2$ and $\pi/4$ on the second and third qubits, respectively.
2. A Hadamard gate on the second qubit, followed by a controlled phase rotation with angle $\pi/2$ on the third qubit.
3. A Hadamard gate on the third qubit.
4. Swapping the qubits to reverse their order (since QFT output is in reverse order).

Qiskit Code

The Qiskit implementation of the 3-qubit QFT is as follows:

```
from qiskit import QuantumCircuit, Aer, execute
import numpy as np
from qiskit.visualization import plot_bloch_multivector

# Function to create a 3-qubit QFT circuit
def qft_3qubit():
    qc = QuantumCircuit(3)

    # Step 1: Apply Hadamard on the first qubit
    qc.h(0)
    qc.cp(np.pi / 2, 2, 1)  # Controlled phase rotation
        between qubits 0 and 1
    qc.cp(np.pi / 4, 2, 0)  # Controlled phase rotation
        between qubits 2 and 0

    # Step 2: Apply Hadamard on the second qubit
    qc.h(1)
    qc.cp(np.pi / 2, 1, 0)  # Controlled phase rotation
        between qubits 1 and 2

```

```
    # Step 3: Apply Hadamard on the third qubit
    qc.h(2)

    # Step 4: Swap qubits to reverse order
    qc.swap(0, 2)

    return qc

# Create and display the QFT circuit for 3 qubits
qc = qft_3qubit()
print("3-qubit QFT circuit:")
display(qc.draw('mpl'))

# Use the statevector simulator to observe the final state
simulator = Aer.get_backend('statevector_simulator')
qc_save_state = qc.copy()  # Copy circuit for observing statevector before measurement
result = execute(qc_save_state, simulator).result()
statevector = result.get_statevector()

# Display the final state vector and plot on Bloch sphere
print("Final state vector:", statevector)
plot_bloch_multivector(statevector)
```

Explanation of the Code

Hadamard and Controlled Phase Rotations

- **First Qubit (Qubit 0):** A Hadamard gate is applied to the first qubit, followed by controlled phase rotations with angles $\pi/2$ and $\pi/4$ on the second and third qubits, respectively. This step introduces phase relationships among the qubits based on their positions.
- **Second Qubit (Qubit 1):** After applying a Hadamard gate to the second qubit, a controlled phase rotation with angle $\pi/2$ is applied between the second and third qubits.
- **Third Qubit (Qubit 2):** A Hadamard gate is applied to the third qubit.

Swapping Qubits

The `qc.swap(0, 2)` operation swaps the first and third qubits to reverse their order. This step is necessary because the QFT output is in reverse order of the input qubits.

Simulation and Visualization

The `statevector_simulator` is used to obtain the final state vector after applying the QFT. The `plot_bloch_multivector` function visualizes the final state on the Bloch sphere, showing the transformations achieved by the QFT.

Explanation of Results

The 3-qubit QFT transforms the quantum state into the Fourier basis, establishing specific phase relationships among the qubits. The final state vector represents this transformed state, and visualizing it on the Bloch sphere reveals the effect of the QFT on a multi-qubit system.

A 4-Qubit QFT

Introduction

For a 4-qubit Quantum Fourier Transform (QFT), we apply a sequence of Hadamard gates and controlled-phase rotations with progressively smaller angles among the qubits. The 4-qubit QFT is an extension of the 2- and 3-qubit QFTs, where each qubit undergoes more interactions to establish phase relationships with the others.

Steps for 4-Qubit QFT

For a 4-qubit system, the QFT consists of the following:

1. A Hadamard gate on the first qubit, followed by controlled-phase rotations with angles $\pi/2$, $\pi/4$, and $\pi/8$ on the second, third, and fourth qubits, respectively.
2. A Hadamard gate on the second qubit, followed by controlled-phase rotations with angles $\pi/2$ and $\pi/4$ on the third and fourth qubits.
3. A Hadamard gate on the third qubit, followed by a controlled-phase rotation with angle $\pi/2$ on the fourth qubit.
4. A Hadamard gate on the fourth qubit.
5. Finally, swap the qubits to reverse their order (since QFT output is in reverse order).

Qiskit Code

The Qiskit code for implementing the 4-qubit QFT is as follows:

```
from qiskit import QuantumCircuit, Aer, execute
import numpy as np
from qiskit.visualization import plot_bloch_multivector

# Function to create a 4-qubit QFT circuit
def qft_4qubit():
    qc = QuantumCircuit(4)

    # Step 1: Apply Hadamard on the first qubit
    qc.h(0)
    qc.cp(np.pi / 2, 3, 2)  # Controlled phase rotation
        between qubits 3 and 2
    qc.cp(np.pi / 4, 3, 1)  # Controlled phase rotation
        between qubits 3 and 1
```

```
    qc.cp(np.pi / 8, 2, 0)  # Controlled phase rotation between qubits 2 and 0

    # Step 2: Apply Hadamard on the second qubit
    qc.h(1)
    qc.cp(np.pi / 2, 2, 0)  # Controlled phase rotation between qubits 2 and 0
    qc.cp(np.pi / 4, 2, 1)  # Controlled phase rotation between qubits 2 and 1

    # Step 3: Apply Hadamard on the third qubit
    qc.h(2)
    qc.cp(np.pi / 2, 2, 3)  # Controlled phase rotation between qubits 2 and 3

    # Step 4: Apply Hadamard on the fourth qubit
    qc.h(3)

    # Step 5: Swap qubits to reverse the order
    qc.swap(0, 3)
    qc.swap(1, 2)

    return qc

# Create and display the QFT circuit for 4 qubits
qc = qft_4qubit()
print("4-qubit QFT circuit:")
display(qc.draw('mpl'))

# Use the statevector simulator to observe the final state
simulator = Aer.get_backend('statevector_simulator')
qc_save_state = qc.copy()  # Copy circuit for observing statevector before measurement
result = execute(qc_save_state, simulator).result()
statevector = result.get_statevector()

# Display the final state vector and plot on Bloch sphere
print("Final state vector:", statevector)
plot_bloch_multivector(statevector)
```

Explanation of the Code

Hadamard and Controlled Phase Rotations

- **First Qubit (Qubit 0):** Apply a Hadamard gate to qubit 0, followed by controlled-phase rotations with angles $\pi/2$, $\pi/4$, and $\pi/8$ on qubits 1, 2, and 3, respectively. This step establishes initial phase relationships with the other qubits.

- **Second Qubit (Qubit 1):** Apply a Hadamard gate to qubit 1, followed by controlled-phase rotations with angles $\pi/2$ and $\pi/4$ on qubits 2 and 3.
- **Third Qubit (Qubit 2):** Apply a Hadamard gate to qubit 2, followed by a controlled-phase rotation with angle $\pi/2$ on qubit 3.
- **Fourth Qubit (Qubit 3):** Apply a Hadamard gate to qubit 3.

Swapping Qubits

The swap operations, `qc.swap(0, 3)` and `qc.swap(1, 2)`, reorder the qubits in reverse order. This is necessary because the QFT output is conventionally represented in reverse bit order.

Simulation and Visualization

The `statevector_simulator` is used to get the final state vector of the qubits after applying the QFT. The `plot_bloch_multivector` function visualizes the final state on the Bloch sphere, showing the effect of the QFT on the qubits.

Explanation of Results

The 4-qubit QFT transforms the 4-qubit system into the Fourier basis, creating complex phase relationships and superpositions. The state vector after the QFT represents this transformed state, and visualizing it on the Bloch sphere helps illustrate the phase relationships established by the QFT.

This code demonstrates how the 4-qubit QFT operates by combining Hadamard gates and controlled-phase rotations, followed by swapping qubits to maintain the correct output order.

A 5-Qubit QFT

Introduction

For a 5-qubit Quantum Fourier Transform (QFT), we apply a sequence of Hadamard gates and controlled-phase rotations with progressively smaller angles among the qubits. The 5-qubit QFT extends the same structure as smaller QFT circuits but with additional controlled-phase rotations to establish phase relationships.

Steps for 5-Qubit QFT

For a 5-qubit system, the QFT consists of:

1. A Hadamard gate on the first qubit, followed by controlled-phase rotations with angles $\pi/2$, $\pi/4$, $\pi/8$, and $\pi/16$ on the subsequent qubits.
2. A Hadamard gate on the second qubit, followed by controlled-phase rotations with angles $\pi/2$, $\pi/4$, and $\pi/8$ on the remaining qubits.
3. A Hadamard gate on the third qubit, followed by controlled-phase rotations with angles $\pi/2$ and $\pi/4$ on the subsequent qubits.
4. A Hadamard gate on the fourth qubit, followed by a controlled-phase rotation with angle $\pi/2$ on the fifth qubit.
5. A Hadamard gate on the fifth qubit.
6. Finally, swap the qubits to reverse their order (since QFT output is in reverse order).

Qiskit Code

The Qiskit code for implementing the 5-qubit QFT is as follows:

```
from qiskit import QuantumCircuit, Aer, execute
import numpy as np
from qiskit.visualization import plot_bloch_multivector

# Function to create a 5-qubit QFT circuit
def qft_5qubit():
    qc = QuantumCircuit(5)

    # Step 1: Apply Hadamard on the first qubit
    qc.h(0)
    qc.cp(np.pi / 2, 0, 1)   # Controlled phase rotation
        with pi/2 between qubits 0 and 1
```

```
    qc.cp(np.pi / 4, 0, 2)   # Controlled phase rotation with pi/4 between qubits 0 and 2
    qc.cp(np.pi / 8, 0, 3)   # Controlled phase rotation with pi/8 between qubits 0 and 3
    qc.cp(np.pi / 16, 0, 4)  # Controlled phase rotation with pi/16 between qubits 0 and 4

    # Step 2: Apply Hadamard on the second qubit
    qc.h(1)
    qc.cp(np.pi / 2, 1, 2)   # Controlled phase rotation with pi/2 between qubits 1 and 2
    qc.cp(np.pi / 4, 1, 3)   # Controlled phase rotation with pi/4 between qubits 1 and 3
    qc.cp(np.pi / 8, 1, 4)   # Controlled phase rotation with pi/8 between qubits 1 and 4

    # Step 3: Apply Hadamard on the third qubit
    qc.h(2)
    qc.cp(np.pi / 2, 2, 3)   # Controlled phase rotation with pi/2 between qubits 2 and 3
    qc.cp(np.pi / 4, 2, 4)   # Controlled phase rotation with pi/4 between qubits 2 and 4

    # Step 4: Apply Hadamard on the fourth qubit
    qc.h(3)
    qc.cp(np.pi / 2, 3, 4)   # Controlled phase rotation with pi/2 between qubits 3 and 4

    # Step 5: Apply Hadamard on the fifth qubit
    qc.h(4)

    # Step 6: Swap qubits to reverse the order
    qc.swap(0, 4)
    qc.swap(1, 3)

    return qc

# Create and display the QFT circuit for 5 qubits
qc = qft_5qubit()
print("5-qubit QFT circuit:")
display(qc.draw('mpl'))

# Use the statevector simulator to observe the final state
simulator = Aer.get_backend('statevector_simulator')
qc_save_state = qc.copy()  # Copy circuit for observing statevector before measurement
result = execute(qc_save_state, simulator).result()
statevector = result.get_statevector()

# Display the final state vector and plot on Bloch sphere
```

```
print("Final state vector:", statevector)
plot_bloch_multivector(statevector)
```

Explanation of the Code

Hadamard and Controlled Phase Rotations

- **First Qubit (Qubit 0):** A Hadamard gate is applied to qubit 0, followed by controlled-phase rotations with angles $\pi/2$, $\pi/4$, $\pi/8$, and $\pi/16$ on the subsequent qubits. This creates phase relationships with the other qubits.
- **Second Qubit (Qubit 1):** A Hadamard gate is applied to qubit 1, followed by controlled-phase rotations with angles $\pi/2$, $\pi/4$, and $\pi/8$ on the remaining qubits.
- **Third Qubit (Qubit 2):** A Hadamard gate is applied to qubit 2, followed by controlled-phase rotations with angles $\pi/2$ and $\pi/4$ on qubits 3 and 4.
- **Fourth Qubit (Qubit 3):** A Hadamard gate is applied to qubit 3, followed by a controlled-phase rotation with angle $\pi/2$ on qubit 4.
- **Fifth Qubit (Qubit 4):** A Hadamard gate is applied to qubit 4.

Swapping Qubits

The `qc.swap(0, 4)` and `qc.swap(1, 3)` operations reorder the qubits in reverse order. This step is necessary as the QFT output is typically in reverse bit order.

Simulation and Visualization

We use the `statevector_simulator` to obtain the final state vector after the QFT. The `plot_bloch_multivector` function visualizes the final state on the Bloch sphere, helping illustrate the phase relationships established by the QFT.

Explanation of Results

The 5-qubit QFT transforms the state of the 5-qubit system into the Fourier basis, establishing specific phase relationships between the qubits. The final state vector after applying the QFT reflects this transformation, and visualizing it on the Bloch sphere provides insight into how the QFT affects the multi-qubit state. This code demonstrates the step-by-step process of applying the QFT on a 5-qubit system in Qiskit.