MDS572B: QUANTUM MACHINE LEARNING

OBSERVATION NOTEBOOK

Name: Neelanjan Dutta

Roll No: 2448040

Class: 5MDS

LAB No. 2: Basic operations on Qubit and measurement on Bloch sphere

Question:

Construct qubits with all simple gates (X, Y, Z and H) and the combined gate.

Objective:

The objective of this lab is to explore the fundamental operations on a single qubit by applying the **Pauli-X, Y, Z**, and **Hadamard gates**. This lab focuses on visualizing the transformation of the qubit's state vector on the **Bloch sphere** for each gate, thereby providing a clear geometric intuition for these core quantum operations. Additionally, the cumulative effect of a sequence of gates is examined to understand how operations are combined.

Draft Plan:

Program description:

This program applies fundamental single-qubit gates (X, Y, Z, H) and a combination of them to a qubit initially in the **|0>** state. The primary goal is to visualize the resulting state vector's position on the Bloch sphere after each operation to understand the geometric effect of each gate.

Program logic:

- **Initialization:** A single-qubit **QuantumCircuit** was created. By default, this qubit initializes in the state **|0>,** which corresponds to a vector pointing to the north pole of the **Bloch sphere**.
- **Gate Application**: For each gate (X, Y, Z, and H), a new, separate circuit was created to isolate its effect. The specific gate **(.x(0), .y(0), etc.)** is applied to the qubit.
- **State virtualization**: The **Statevector.from_instruction()** method was used to calculate the final quantum state vector after the gate is applied and **plot_bloch_multivector()** takes this state vector as input and plots it as an arrow on the 3D Bloch sphere.

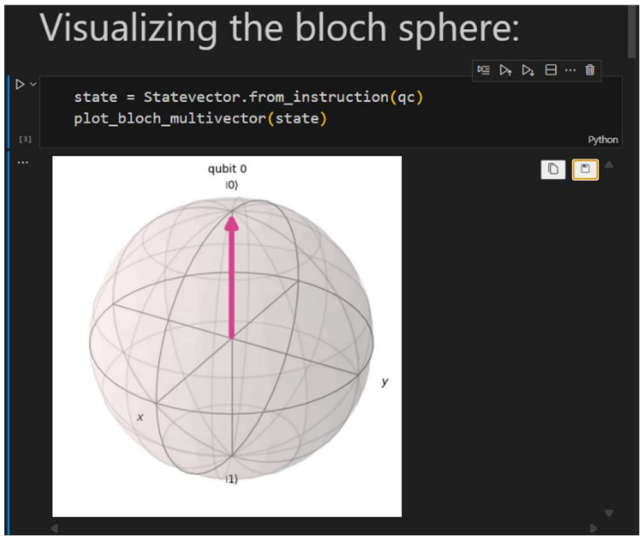
- **Combined Gates**: A final circuit demonstrates the cumulative effect of operations by applying a sequence of gates **(X → Y → Z → H)** before calculating and visualizing the single final state.

Program:

Loading the important libraries:

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_bloch_multivector, plot_histogram
from qiskit.quantum_info import Statevector
import matplotlib.pyplot as plt
```

Visualizing the Bloch sphere:



X-Gate:

The Pauli-X gate, commonly called the X gate, is the quantum equivalent of a classical **NOT** gate. Its primary function is to flip the state of a qubit.

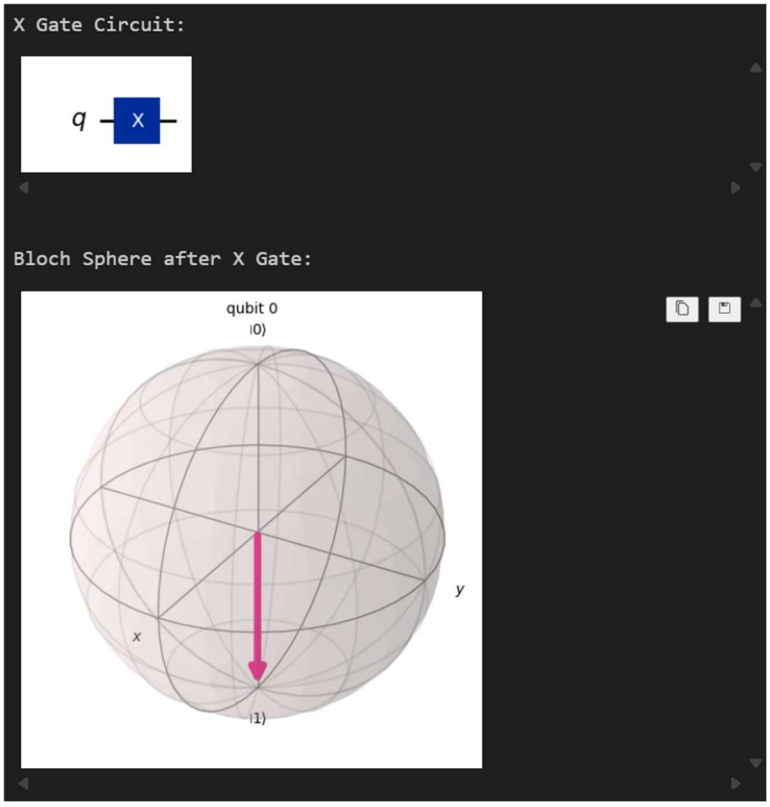
- **Bit-Flip Operation**: It changes the basis state **|0>** to **|1>**, and it changes **|1>** back to **|0>**.
- **Bloch Sphere Rotation:** Geometrically, it represents a **180-degree rotation** of the qubit's state vector around the x-axis of the Bloch sphere. This is why a qubit at the north pole **(|0>)** flips to the south pole **(|1>).**

➢ **Matrix Form**: Mathematically, it's represented by the Pauli-X matrix:

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

```python
# Create a quantum circuit for the X gate
qc_x = QuantumCircuit(1)

# Apply the X gate to the first qubit (at index 0)
qc_x.x(0)

# Draw the circuit
print("X Gate Circuit:")
display(qc_x.draw('mpl'))

# Visualize the resulting state on the Bloch sphere
print("\nBloch Sphere after X Gate:")
state_x = Statevector.from_instruction(qc_x)
display(plot_bloch_multivector(state_x))
```

Output:



Interpretation:

The Bloch sphere in the image provides a geometric representation of the qubit's quantum state. The pink arrow, known as the state vector, points directly downwards from the center to the south pole of the sphere. This specific position corresponds to the definite state. As the title indicates, this is the state *after* an X gate has been applied. This visualization confirms the function of the X gate as a bit-flip, successfully rotating the qubit from its initial state (at the north pole) to the final state.

Y-Gate:

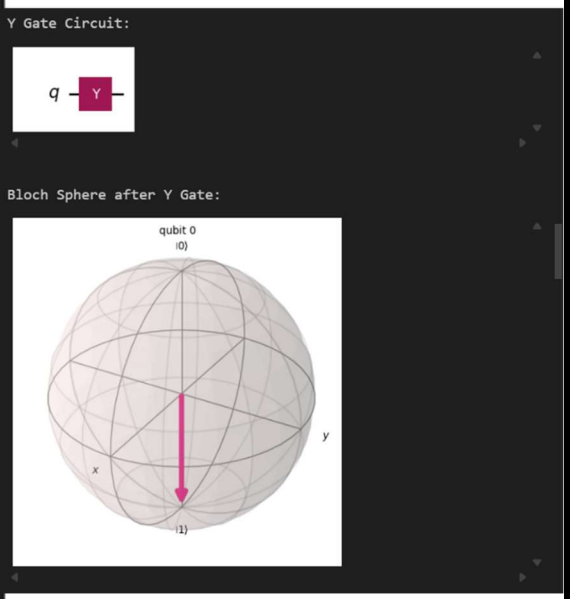
The Pauli-Y gate is a quantum gate that performs both a bit-flip and a phase-flip on a qubit.

- **Bit-Flip Operation**: It changes **|0>** to **i|1>**, and changes **|1>** to **i|0>**.
- **Bloch Sphere Rotation:** Geometrically, it represents a **180-degree rotation** of the qubit's state vector around the y-axis of the Bloch sphere.
- **Matrix Form**: Mathematically, it's represented by the Pauli-Y matrix:

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

Output:

```python
# Create a quantum circuit for the Y gate
qc_y = QuantumCircuit(1)

# Apply the Y gate
qc_y.y(0)

# Draw the circuit
print("Y Gate Circuit:")
display(qc_y.draw('mpl'))

# Visualize the resulting state
print("\nBloch Sphere after Y Gate:")
state_y = Statevector.from_instruction(qc_y)
display(plot_bloch_multivector(state_y))
```



Interpretation:

The Bloch sphere in the image provides a geometric representation of the qubit's quantum state. The pink arrow, known as the state vector, points directly downwards from the center to the south pole of the sphere. This specific position corresponds to the definite state **|1>**. As the title indicates, this is the state after a **Y gate** has been applied. This visualization confirms that the Y gate also acts as a bit-flip, successfully rotating the qubit from its initial **|0>** state (at the north pole) to the final **|1>** state.

Z-Gate:

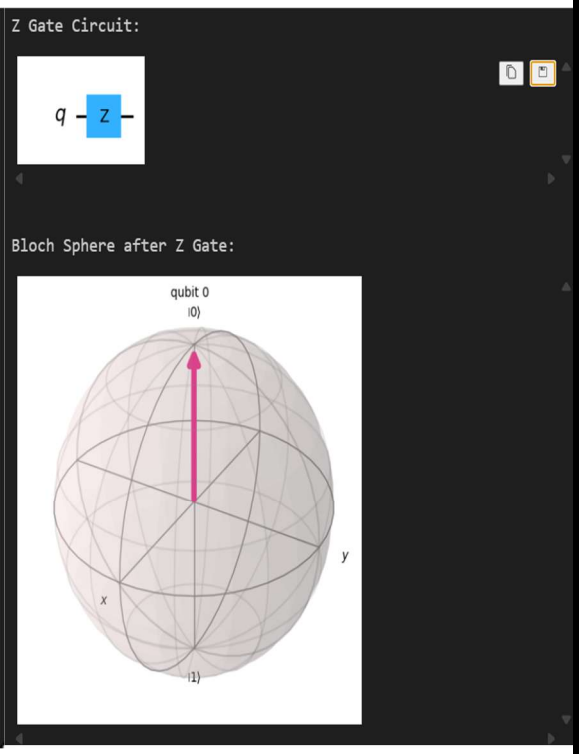
The Pauli-Z gate is commonly known as the phase-flip gate. It alters the phase of a qubit without changing its basis state.

- **Phase-Flip Operation**: It leaves the base state **|0>** unchanged, but it flips the sign (applies a phase of -1) to the |1> state, making it -**|1>**.
- **Bloch Sphere Rotation:** Geometrically, it represents a **180-degree rotation** of the qubit's state vector around the z-axis of the Bloch sphere. This is why it has no visible effect on states |0> or |1>, as they lie on the axis of rotation.
- **Matrix Form**: Mathematically, it's represented by the Pauli-Z matrix:

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

Output:

```python
# Create a quantum circuit for the Z gate
qc_z = QuantumCircuit(1)

# Apply the Z gate
qc_z.z(0)

# Draw the circuit
print("Z Gate Circuit:")
display(qc_z.draw('mpl'))

# Visualize the resulting state
print("\nBloch Sphere after Z Gate:")
state_z = Statevector.from_instruction(qc_z)
display(plot_bloch_multivector(state_z))
```



Interpretation:

The Bloch sphere in the image shows the qubit's state after a Z gate has been applied. The state vector points directly upwards to the north pole, which corresponds to the definite state |0>. This visualization confirms that applying a Z gate to a qubit already in the |0> state has no visible effect, as the Z gate performs a rotation around the z-axis, and the state vector is already on that axis of rotation.

Hadamard-Gate (H-Gate):

The Hadamard gate is a fundamental quantum gate used to create superposition. It's one of the most important gates for building quantum algorithms.

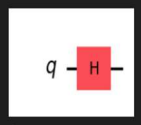
- **Superposition Operation**: It transforms the definite basis states into equal superposition states.
- **Bloch Sphere Rotation:** Geometrically, it represents a **180-degree rotation** around an axis located exactly between the x- and z-axes.
- **Matrix Form**: Mathematically, it's represented by the Hadamard matrix:

$$\left(\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\right)$$

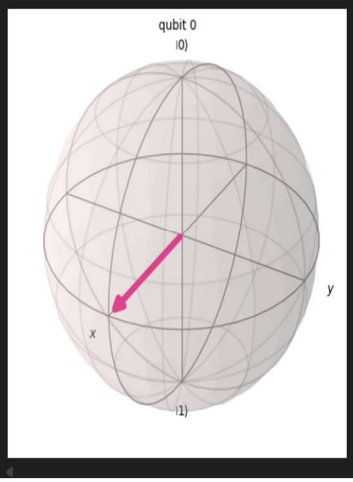
Output

```python
# Create a quantum circuit for the H gate
qc_h = QuantumCircuit(1)

# Apply the H gate
qc_h.h(0)

# Draw the circuit
print("H Gate Circuit:")
display(qc_h.draw('mpl'))

# Visualize the resulting state
print("\nBloch Sphere after H Gate:")
state_h = Statevector.from_instruction(qc_h)
display(plot_bloch_multivector(state_h))
```

```
H Gate Circuit:
```




```
Bloch Sphere after H Gate:
```



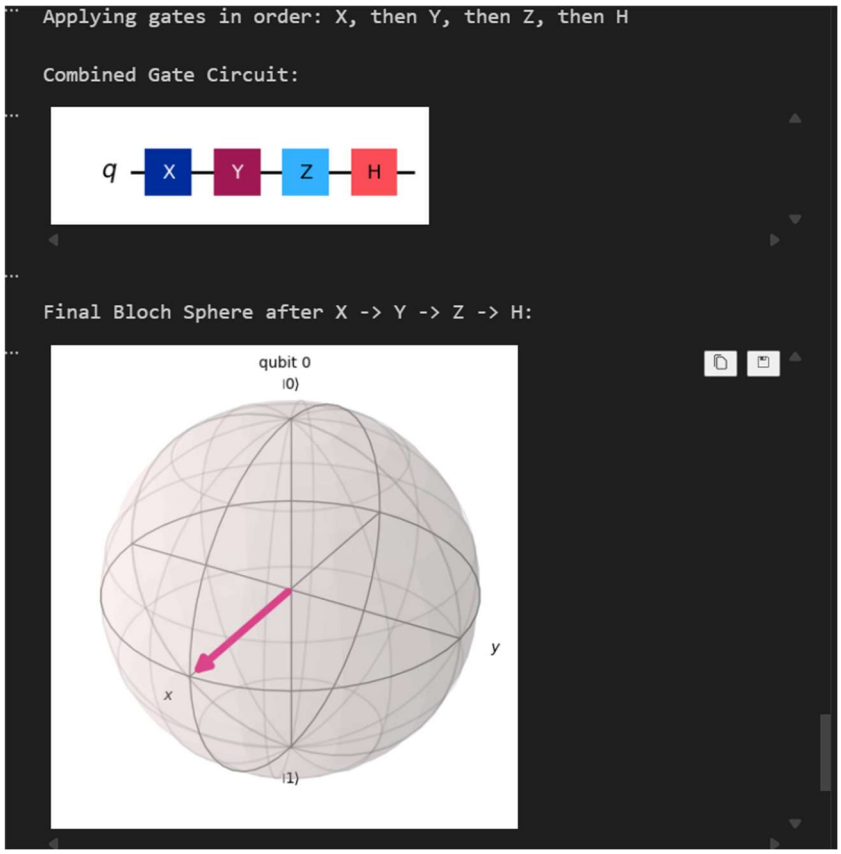
Interpretation:

The Bloch sphere in the image provides a geometric representation of the qubit's quantum state after a Hadamard (H) gate has been applied. The state vector points from the center to the equator of the sphere, directly along the positive x-axis. This position represents an equal superposition of the **|0>** and **|1>** states, known as the |+> state. This visualization confirms the function of the H gate, which is to transform a definite state into a superposition by rotating it from the pole to the equator.

Combined gates:

```python
# Create a circuit for the combined gates
qc_all = QuantumCircuit(1)

# Apply the gates in the specified order: X -> Y -> Z -> H
print("Applying gates in order: X, then Y, then Z, then H")
qc_all.x(0)  # First, apply X gate
qc_all.y(0)  # Second, apply Y gate
qc_all.z(0)  # Third, apply Z gate
qc_all.h(0)  # Finally, apply H gate

# Draw the final circuit
print("\nCombined Gate Circuit:")
display(qc_all.draw('mpl'))

# Visualize the final state on the Bloch sphere
print("\nFinal Bloch Sphere after X -> Y -> Z -> H:")
final_state = Statevector.from_instruction(qc_all)
display(plot_bloch_multivector(final_state))
```

Output:



Interpretation:

The Bloch sphere in the image shows the final state of a qubit after a sequence of four gates (X, Y, Z, and H) has been applied. The state vector points to the equator along the positive x-axis, which corresponds to the equal superposition state known as |+>. This result demonstrates the cumulative effect of the gates: the initial X and Y gates effectively cancel each other out (flipping), the Z gate has no visible effect on the |0> state, and the final Hadamard gate rotates this resulting state into the superposition.

Conclusion:

In conclusion, this lab successfully demonstrated the effects of single-qubit Pauli and Hadamard gates on a qubit initialized in the |0> state. Through Bloch sphere visualizations, the rotational nature of these gates was confirmed: the X and Y gates perform bit-flips, the Z gate induces a phase shift, and the Hadamard gate creates a superposition state. The final test case with combined gates correctly showed their sequential application. This exercise solidifies the understanding of basic qubit manipulation, which is essential for building more complex quantum algorithms

****