

Q3-2



QX03-PhaseAndBlochSphere.ipynb
File

Necesito que me expliques detalladamente cada una de las celdas del siguiente proyecto

A continuación tienes una explicación paso a paso de cada celda de tu notebook **QX03-PhaseAndBlochSphere.ipynb**. He numerado las celdas según aparecen y, para cada una, incluyo:

1. El código completo de la celda.
2. Descripción línea por línea de lo que hace.
3. Conceptos teóricos asociados.

Celda 1 (importaciones y configuraciones iniciales)

python

```
from qiskit import *
from qiskit.tools.visualization import plot_bloch_multivector
from qiskit.visualization import plot_histogram
%matplotlib inline
import math
```

- **from qiskit import ***: importa todas las clases y funciones principales de Qiskit (QuantumCircuit, Aer, execute, etc.).
- **from qiskit.tools.visualization import plot_bloch_multivector**: trae la función para dibujar en una esfera de Bloch el vector de estado completo de un registro cuántico.
- **from qiskit.visualization import plot_histogram**: importa la herramienta para representar histogramas de conteos de medición.
- **%matplotlib inline**: instrucción mágica de Jupyter para incrustar las gráficas de Matplotlib directamente en la salida de la celda.
- **import math**: importa el módulo estándar de Python con funciones matemáticas (por ejemplo, `math.pi`).

Teoría:

- **Esfera de Bloch**: representación geométrica del estado de un solo qubit como un punto sobre la superficie de una esfera unitaria.
- **Histogramas de Qiskit**: muestran la distribución de resultados de medir un circuito cuántico en el backend de tipo "qasm".

Celda 2 (selección de backends de simulación)

python

```
qasm_simulator = Aer.get_backend('qasm_simulator')
statevector_simulator = Aer.get_backend('statevector_simulator')
```

- `Aer.get_backend('qasm_simulator')`: crea un simulador que emula mediciones reales (genera muestras de resultados discretos).
- `Aer.get_backend('statevector_simulator')`: crea un simulador que devuelve el estado cuántico completo (vector de amplitudes) antes de medir.

Teoría:

- El **simulador de vectores de estado** devuelve amplitudes complejas $|\psi\rangle$.
- El **simulador QASM** modela la estadística de medidas («shots»), devolviendo conteos de cada bitstring.

Celda 3 (definición de función de ejecución)

python

```
def run_on_simulators(circuit):
    # 1) Ejecuta para obtener el vector de estado
    statevec_job = execute(circuit, backend=statevector_simulator)
    result = statevec_job.result()
    statevec = result.get_statevector()

    # 2) Mide todos los qubits en bits clásicos
    num_qubits = circuit.num_qubits
    circuit.measure([i for i in range(num_qubits)], [i for i in range(num_qubits)])

    # 3) Ejecuta simulación QASM para conteos
    qasm_job = execute(circuit, backend=qasm_simulator, shots=1024).result()
    counts = qasm_job.get_counts()

    return statevec, counts
```

- **Línea 2-4:** ejecuta el circuito sin medirlo, obtiene el vector de estado completo.
- **Línea 6-7:** añade operaciones de medición (cada qubit \rightarrow su bit clásico).
- **Línea 9-10:** ejecuta 1024 tiros ("shots") para simular estadísticas de medida.
- **Retorno:**
 - `statevec`: lista de amplitudes complejas α_i .
 - `counts`: diccionario {bitstring: número_de_ocurrencias}.

Teoría:

- La **medición** colapsa el estado cuántico: por eso se ejecuta de nuevo tras obtener el estado puro.
- Se usan dos simulaciones distintas para separar la **visualización continua** (estado puro) y la **estadística de mediciones**.

Celda 4 (estado inicial $|00\rangle$)

python

```
circuit = QuantumCircuit(2,2)
statevec, counts = run_on_simulators(circuit)
```

- Instancia un circuito con **2 qubits** y **2 bits clásicos**.
- Sin aplicar puertas, el estado por defecto es $|00\rangle$.
- Llama a `run_on_simulators` para obtener vector y conteos.

Teoría:

- Estado por defecto en Qiskit es $|0\rangle$ para cada qubit.
- Vector de estado resultante: $[1, 0, 0, 0]$ en la base $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$.

Celda 5 (Bloch de cada qubit en $|00\rangle$)

python

```
plot_bloch_multivector(statevec)
```

- Dibuja la esfera de Bloch para **cada uno de los dos qubits**, mostrando sus vectores reducidos.

Teoría:

- En $|0\rangle$, cada qubit apunta al **polo norte** ($\theta = 0$, ϕ indefinido).
- La función extrae el **estado reducido** de cada qubit y lo proyecta.

Celda 6 (superposición simple)

python

```
circuit.h(0)
statevec, counts = run_on_simulators(circuit)
plot_bloch_multivector(statevec)
```

1. `circuit.h(0)`: aplica la puerta de **Hadamard** al qubit 0, generando $(|0\rangle + |1\rangle)/\sqrt{2}$.
2. Ejecuta y dibuja la nueva posición sobre la esfera de Bloch.

Teoría:

- La puerta H rota θ a $\pi/2$ y orienta $\phi = 0$: coloca el vector en el **ecuador** hacia el eje X+.

Celda 7 (entrelazamiento / estado de Bell)

python

```
circuit = QuantumCircuit(2,2)
circuit.h(0)
circuit.cx(0,1)
statevec, counts = run_on_simulators(circuit)
plot_bloch_multivector(statevec)
```

1. Nuevo circuito de 2 qubits.
2. **H en qubit 0** → superposición.
3. **CX(0→1)** → crea la **paridad**: estado

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

4. Se grafica la esfera de Bloch de cada qubit por separado.

Teoría:

- Aunque el par de qubits esté **entrelazado**, al mirar cada qubit individualmente se ve un estado **mezcla** en el centro de la esfera (vector nulo).

Celda 8 (histograma del estado de Bell)

python

```
plot_histogram([counts])
```

- Representa la distribución de mediciones: debería mostrar **~50% 00**, **~50% 11**.

Teoría:

- El histograma refleja la **probabilidad** de cada bitstring tras medir.

Celda 9 (rotaciones en X)

python

```
circuit = QuantumCircuit(2,2)
circuit.rx(math.pi/4, 0)
circuit.rx(math.pi/2, 1)
statevec, counts = run_on_simulators(circuit)
plot_bloch_multivector(statevec)
```

1. **rx(θ, q)**: rota el qubit q alrededor del eje X un ángulo θ.
 - Qubit 0: $\theta = \pi/4$.
 - Qubit 1: $\theta = \pi/2$.
2. Muestra las posiciones en la esfera.

Teoría:

- Gate RX es equivalente a $e^{-i\theta X/2}$.
- Sobre la Bloch, significa girar desde el polo norte hacia el ecuador X.

Celda 10 (rotaciones en Y)

```
python

circuit = QuantumCircuit(2,2)
circuit.ry(math.pi/4, 0)
circuit.ry(math.pi/2, 1)
statevec, counts = run_on_simulators(circuit)
plot_bloch_multivector(statevec)
```

- Igual que la anterior, pero **RY** rota alrededor del eje Y.
- Se observa el vector en un plano perpendicular al de RX.

Teoría:

- Gate RY: $e^{-i\theta Y/2}$.
- Cambia el ángulo polar θ sin alterar φ .

Celda 11 (superposición y Bloch en 1 qubit)

```
python

circuit = QuantumCircuit(1,1)
circuit.h(0)
statevec, counts = run_on_simulators(circuit)
plot_bloch_multivector(statevec)
```

- Misma idea que en la celda 6, pero con **un solo qubit**.
- Útil para centrar la explicación en la esfera de un qubit.

Celda 12 (fase con puerta Z)

```
python

circuit = QuantumCircuit(1,1)
circuit.h(0)
circuit.z(0)
statevec, counts = run_on_simulators(circuit)
plot_bloch_multivector(statevec)
```

1. **H(0)**: lleva $|0\rangle$ al eje X+.
2. **Z(0)**: aplica la puerta de fase Z:

$$|0\rangle \mapsto |0\rangle, \quad |1\rangle \mapsto -|1\rangle.$$

3. El efecto en la esfera es una **rotación de 180°** alrededor del eje Z (cambia φ en π).

Teoría:

- Las puertas Z no cambian el valor de medición en la base Z, pero modifican la fase relativa.
- En la esfera, esto equivale a rotar en el plano XY.

Celda 13 (histograma final)

```
python
plot_histogram([counts])
```

- Muestra la distribución de mediciones del experimento con puerta Z.
- Como Z no afecta las probabilidades en la base Z, deberías ver **100%** $|+\rangle$ medido \rightarrow tras H+Z vuelta a $|-\rangle$ en la base X, pero en Z la distribución sigue siendo 50/50 si mides en Z tras H alone. (Revisar contexto de medida.)

Celda 14 (vacía)

- No contiene código ni comentarios. Puede eliminarse o usarse para anotaciones.

Resumen de conceptos clave

- **QuantumCircuit(n,m)**: crea un circuito con n qubits y m bits clásicos.
- **Puertas H, RX, RY, Z, CX**: rotaciones y entrelazamiento.
- **Simuladores Aer**: `statevector_simulator` para vectores puros, `qasm_simulator` para conteos.
- **Esfera de Bloch**: visualiza la orientación del vector de estado de un solo qubit.
- **Histogramas**: muestran probabilidades de resultados medidos.

Con esto deberías tener claro qué hace cada celda, línea a línea, y la teoría cuántica y de Qiskit subyacente. ¡Avísame si necesitas más detalle sobre algún punto en particular!