

**Arquitecturas y Programación de  
Computadores Cuánticos**  
**Práctica 1: Entorno de trabajo, puertas  
cuánticas, superposición y entrelazamiento**

*Prof. Alberto A. del Barrio*

*Prof. Guillermo Botella*

## Índice

<b>1. Objetivos</b>	<b>3</b>
<b>2. Entorno de trabajo</b>	<b>3</b>
<b>3. Superposición</b>	<b>4</b>
<b>4. Entrelazamiento</b>	<b>4</b>
4.1. Los estados de Bell . . . . .	4
4.2. El estado GHZ . . . . .	5
<b>5. Simulación y backends</b>	<b>5</b>
<b>6. Desarrollo de la práctica</b>	<b>7</b>
6.1. Parte guiada . . . . .	7
6.2. Parte no guiada . . . . .	9

## 1. Objetivos

El componente principal que vamos a utilizar en el laboratorio es una máquina virtual con un Ubuntu 18.04-LTS. La instalación de los diversos frameworks está basada en Anaconda [1] y en sus entornos. Para ejecutar los códigos, utilizaremos el editor interactivo Jupyter Notebook [3]. En esta primera práctica trataremos de familiarizarnos con el entorno de trabajo y empezar a programar circuitos cuánticos sencillos. Por tanto, los objetivos de esta práctica son:

- Familiarizarse con el entorno de desarrollo y comparar los resultados obtenidos con distintos frameworks.
- Realizar simulaciones y ejecuciones en backends reales.
- Familiarizarse con las puertas cuánticas básicas.
- Entender los conceptos cuánticos de superposición y entrelazamiento.

## 2. Entorno de trabajo

Como hemos mencionado anteriormente, el entorno de trabajo básico será una máquina virtual con un Ubuntu 18.04-LTS. Dado que todos los frameworks de programación que utilizaremos están basados en Python, haremos uso de Anaconda [1] para facilitar la instalación y gestión de los mismos. Anaconda es una distribución open-source de Python (y R) que contiene numerosas librerías. Una de sus mayores ventajas es el uso de los entornos, aka *environments*. Con Anaconda, se pueden crear, exportar, enumerar, eliminar y actualizar entornos que tienen diferentes versiones de Python y/o paquetes instalados en ellos. Cambiar o moverse entre entornos se denomina activación del entorno.

En primer lugar, listaremos todos los entornos instalados a través del comando `conda env list`. Deberíamos obtener una lista de 5 entornos además del *base*: *qiskitEnv*, *cirqEnv*, *forestEnv*, *projectqEnv* y *tketEnv*. Cada uno de estos entornos se corresponde con los frameworks que hemos estudiado en clase, respectivamente: *Qiskit* (IBM), *Cirq* (Google), *Forest* (Rigetti), *ProjectQ* (ETH Zurich) y *tket* (Cambridge Quantum Computing).

Aunque la máquina virtual dispone de distintos entornos instalados, la mayor parte de las prácticas las explicaremos a través de *Qiskit*, ya que gráficamente es el entorno más potente.

### 3. Superposición

Tal y como hemos mencionado en clase, los qubits pueden encontrarse en *superposición*. En otras palabras, *ser 0 y 1 al mismo tiempo*. Un modo sencillo de poner un qubit en superposición es aplicar una puerta Hadamard, como veremos en la Sección 6.1.

### 4. Entrelazamiento

Otro de los fenómenos cuánticos fundamentales es el *entrelazamiento*. Los qubits pueden estar entrelazados, esto es, el valor obtenido tras la medición de un qubit determina el valor del otro, *aunque estuvieran a años luz de distancia*.

#### 4.1. Los estados de Bell

En primer lugar, este fenómeno lo estudiaremos por medio de los estados de Bell. El más sencillo de los 4 estados de Bell se consigue aplicando los siguientes pasos:

- Creamos un circuito con 2 qubits.
- Aplicamos la puerta Hadamard sobre uno de ellos.
- Aplicamos una puerta CNOT donde el qubit de control será la salida de la Hadamard, y el qubit objetivo será el otro qubit del circuito (qubit 1).
- Medimos ambos qubits.

En este caso observaremos cómo las salidas son siempre 00 o 11, pero nunca 01 ni 10. Además de este estado, conocido como  $|\Phi^+\rangle$ , existen otros 3. Los 4 estados de Bell son estados *entrelazados al máximo* (*maximally entangled states*) y se definen de acuerdo a las Ecuaciones (1)-(4). Tienen la particularidad de que además forman una base.

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |0\rangle_B + |1\rangle_A \otimes |1\rangle_B) , \quad (1)$$

$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |0\rangle_B - |1\rangle_A \otimes |1\rangle_B) , \quad (2)$$

$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |1\rangle_B + |1\rangle_A \otimes |0\rangle_B) , \quad (3)$$

$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |1\rangle_B - |1\rangle_A \otimes |0\rangle_B) . \quad (4)$$

## 4.2. El estado GHZ

Los 4 estados de Bell pueden generalizarse para más qubits. En concreto, estudiaremos el estado GHZ (Greenberger-Horne-Zeillinger), un estado entrelazado al máximo en un sistema de 3 qubits.

Para generar el estado GHZ basta con añadir un nuevo qubit al estado de Bell mencionado en la Sección 4.1 y añadir una puerta CNOT que utilice el qubit 1 como qubit de control y el qubit 2 como qubit objetivo.

## 5. Simulación y backends

Como hemos mencionado en clase, es posible simular y ejecutar circuitos cuánticos. En esta práctica estudiaremos cómo simular y ejecutar circuitos cuánticos utilizando el framework Qiskit.

En el caso de la ejecución en un computador cuántico de IBM, lo primero será disponer de una cuenta en IBM Quantum Experience y conseguir un token que copiaremos en nuestro código [4]. Una vez accedamos a nuestra cuenta, podemos observar los computadores cuánticos disponibles, así como datos sobre su volumen cuántico, tasa de errores, mapa de conexión de qubits, etc. e incluso cantidad de trabajos lanzados en ellos (ver Figura 1). El esquema de código a seguir se encuentra a continuación.

Código 1: Código para ejecutar un programa basado en Qiskit en un computador cuántico real de IBM

```
from qiskit import IBMQ, assemble, transpile
IBMQ.save_account('MY_TOKEN')

#####
5 # Crear circuito #
#####

provider = IBMQ.load_account()
backend = provider.backends.ibmq_vigo
10 qobj = assemble(transpile(circuit, backend=backend), backend=backend)
    job = backend.run(qobj)
    retrieved_job = backend.retrieve_job(job.job_id())

#####
15 # Mostrar resultados #
#####
```

Para ejecutar una simulación con ruido, hemos de descargarnos el modelo de ruido de un backend real. En este caso utilizaremos el modelo del *ibmq\_vigo*, computador de 5-qubits con un volumen cuántico de 16 (el mismo que hemos utilizado en el ejemplo de ejecución real). El esquema de código a seguir se encuentra a continuación.

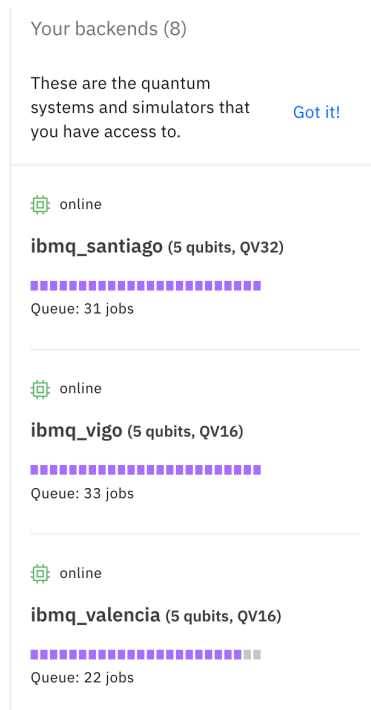


Figura 1: Backends disponibles a través de IBM Quantum Experience

Código 2: Código para simular un programa basado en Qiskit con un modelo de ruido

```
from qiskit import QuantumCircuit, execute
from qiskit import IBMQ, Aer
from qiskit.visualization import plot_histogram
from qiskit.providers.aer.noise import NoiseModel
5
# Construir un modelo de ruido a partir de las características de un
# backend real
provider = IBMQ.load_account()
backend = provider.get_backend('ibmq_vigo')
noise_model = NoiseModel.from_backend(backend)
10
# Obtener el mapa de interconexión de los qubits
coupling_map = backend.configuration().coupling_map

# Obtener las características de las puertas básicas
15
basis_gates = noise_model.basis_gates

#####
# Crear circuito #
#####
20
# Perform a noise simulation
result = execute(circ, Aer.get_backend('qasm_simulator'),
                 coupling_map=coupling_map,
                 basis_gates=basis_gates,
25
                 noise_model=noise_model).result()

#####
# Mostrar resultados #
#####
```

La Figura 2 muestra los histogramas correspondientes a la ejecución del circuito GHZ en

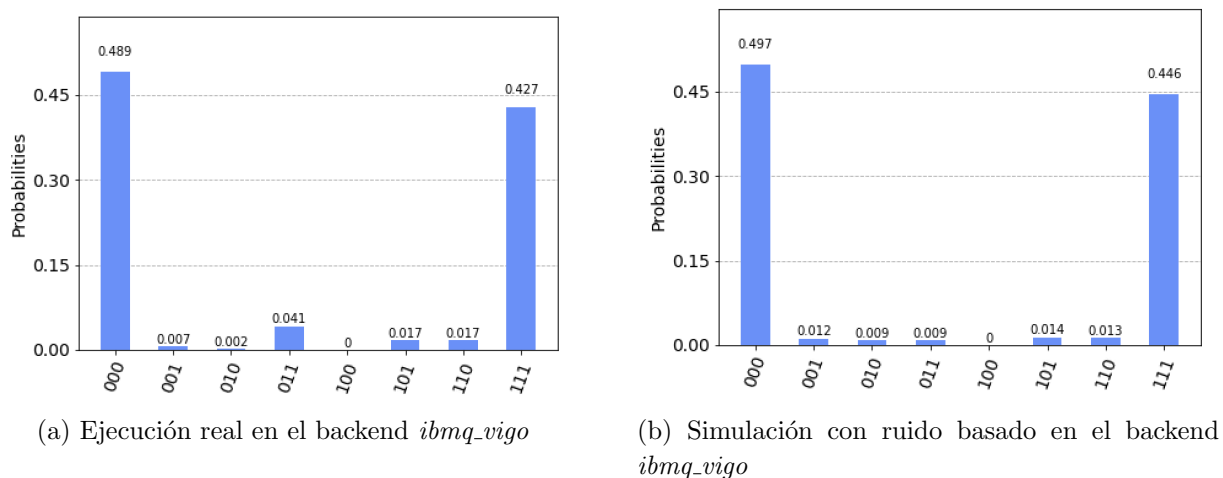


Figura 2: Ejecución y simulación con ruido del circuito GHZ

el backend *ibmq\_vigo* y la la simulación con el modelo de ruido de dicho backend. Como podemos observar, tras 1000 shots, los resultados son bastante parecidos.

## 6. Desarrollo de la práctica

La práctica está dividida en dos partes. Una primera parte guiada, en la que nos familiarizaremos con el entorno de desarrollo. Y una segunda parte en la que el alumno tendría que realizar ciertos ejercicios.

### 6.1. Parte guiada

En esta parte guiada introduciremos el uso de Jupyter a través de un circuito sencillo basado en Qiskit. En primer lugar, hemos de activar el entorno que vayamos a utilizar, en este caso Qiskit, por medio del comando `conda activate qiskitEnv`. Una vez dentro del entorno, por ejemplo podemos listar los paquetes instalados por medio del comando `conda list`. Si tuviéramos permisos, podríamos instalar/desinstalar paquetes a través de la herramienta `pip`. Podéis consultar más comandos en la guía de usuario de Anaconda [2].

Una vez nos encontremos en el entorno de Qiskit, nos moveremos al directorio `$HOME/laboratorios` y crearemos el directorio `p1` y, a su vez, el subdirectorio `qiskit`. Para mantener una mayor organización, dentro del directorio asignado a cada práctica, crearemos un subdirectorio por entorno utilizado.

Dentro del directorio `$HOME/laboratorios/p1/qiskit`, lanzaremos Jupyter Notebook por medio del comando `jupyter notebook`. Se abrirá un navegador y crearemos un nuevo documento tras pulsar *New*  $\rightarrow$  *Python3*.

En esta práctica crearemos un generador de números aleatorios donde la salida puede ser 0 o 1. En Computación Cuántica, basta con poner un qubit en superposición mediante una puerta Hadamard y medir la salida para que colapse el qubit a 0 o a 1. Dicho código se encuentra a continuación.

Código 3: Código generador de números aleatorios en Qiskit

```
import numpy as np
from qiskit import (
    QuantumCircuit,
    execute,
    Aer)
5 from qiskit.visualization import plot_histogram

# Usamos el qasm_simulator de Aer
simulator = Aer.get_backend('qasm_simulator')
10 # Creamos un objeto Quantum Circuit que actúa sobre el registro cuántico
    por defecto (q)
    # de un bit (primer parámetro) y que tiene un registro clásico de un bit
        (segundo parámetro)
circuit = QuantumCircuit(1, 1)
# Añadimos una puerta Hadamard con el qubit q_0 como entrada
circuit.h(0)
15 # Mapeamos la medida de los qubits (primer parámetro) sobre los bits
    clásicos
circuit.measure([0], [0])
# Ejecutamos el circuito sobre el simulador qasm
job = execute(circuit, simulator, shots=1000)
# Almacenamos los resultados
20 result = job.result()
# Capturamos las ocurrencias de salida
counts = result.get_counts(circuit)
# Escribimos el número de ocurrencias
print("\nNúmero de ocurrencias para 0 y 1:", counts)
25 # Dibujamos el circuito
circuit.draw()
```

La Figura 3 muestra su ejecución en el cuaderno Jupyter. Para ello basta con seleccionar el cuadro de texto en el que aparece el código y pulsar el botón de *Run*. La salida aparece en la parte inferior de la Figura 3. Como puede observarse, Qiskit permite escribir por pantalla mediante el comando `print`, así como dibujar el circuito en modo texto, algo que veremos en otros entornos como Cirq.

En otro cuadro de texto dentro del cuaderno Jupyter introduciremos el comando `plot_histogram(counts)`. Ejecutando dicho código podremos observar el histograma que aparece en la Figura 4. Si estudiamos el código a fondo, podemos ver cómo hemos especificado que el número de *shots*, aka número de ejecuciones, sea 1000. Es decir, en promedio deberíamos obtener unos 500 casos de salidas a 0 y otros tantos a 1. Es interesante ver que si reejecutamos la orden `plot_histogram(counts)` el resultado se mantiene. Para poder ver un histograma diferente, hay que ejecutar el código de la Figura 3 primeramente.



## 6.2. Parte no guiada

En esta segunda parte de la práctica, se proponen varios ejercicios.

**Ejercicio 1.** Realizar la práctica guiada utilizando los otros entornos disponibles (salvo la parte dedicada al histograma). Comentar similitudes y diferencias.

**Ejercicio 2.** Programar el estado de Bell comentado en la Sección 4.1 utilizando Qiskit. Emplear como backends los 3 simuladores disponibles en Qiskit Aer. Qué diferencias existen?

**Ejercicio 3.** Programar el estado GHZ comentado en la Sección 4.2 utilizando Qiskit. Simularlo con ruido y utilizando un computador cuántico real. Plotear los resultados y comentar las diferencias.

**Ejercicio 4.** Ejecutar el anterior programa sobre distintos computadores reales de IBM. Plotear los resultados y comentar las diferencias.

**Entregables:** todos los códigos desarrollados en formato texto, así como una memoria con respuestas justificadas a los ejercicios propuestos.

## Referencias

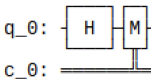
- [1] Anaconda. Accesible en <https://www.anaconda.com/>
- [2] Anaconda Commands. Accesible en <https://docs.conda.io/projects/conda/en/latest/user-guide/index.html>
- [3] Jupyter Notebook. Accesible en <https://jupyter.org/>
- [4] Acceso a máquinas de IBM. Accesible en <https://qiskit.org/documentation/install.html#access-ibm-quantum-systems>

```
In [11]: import numpy as np
from qiskit import(
    QuantumCircuit,
    execute,
    Aer)
from qiskit.visualization import plot_histogram

# Usamos el qasm_simulator de Aer
simulator = Aer.get_backend('qasm_simulator')
# Creamos un objeto Quantum Circuit que actúa sobre el registro cuántico por defecto (q)
# de un bit (primer parámetro) y que tiene un registro clásico de un bit (segundo parámetro)
circuit = QuantumCircuit(1, 1)
# Añadimos una puerta Hadamard con el qubit q_0 como entrada
circuit.h(0)
# Mapeamos la medida de los qubits (primer parámetro) sobre los bits clásicos
circuit.measure([0], [0])
# Ejecutamos el circuito sobre el simulador qasm
job = execute(circuit, simulator, shots=1000)
# Almacenamos los resultados
result = job.result()
# Capturamos las ocurrencias de salida
counts = result.get_counts(circuit)
# Escribimos el número de ocurrencias
print("\nNúmero de ocurrencias para 0 y 1:",counts)
# Dibujamos el circuito
circuit.draw()
```

Número de ocurrencias para 0 y 1: {'0': 479, '1': 521}

Out[11]:



```
q_0: --[H]--[M]--
      |
c_0:  ==
```

Figura 3: Circuito correspondiente al generador de números aleatorios

```
In [12]: # Mostramos el histograma
plot_histogram(counts)
```

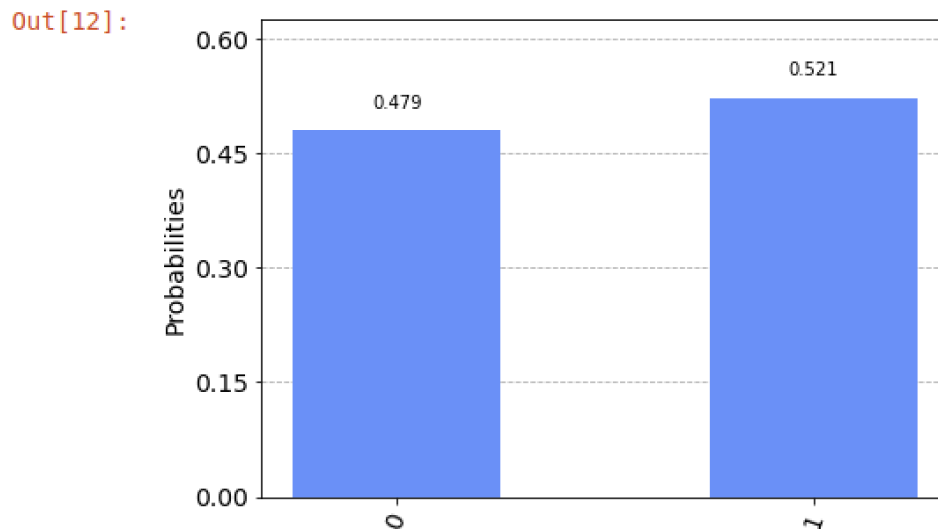


Figura 4: Histograma correspondiente al generador de números aleatorios