Lab_5

December 12, 2024

1 Introduction to Quantum Information and Quantum ML

Instructor: Dr Sci. Eng. Przemysław Głowacki

Kacper Dobek 148247

My solution was inspired by this tutorial and this tutorial.

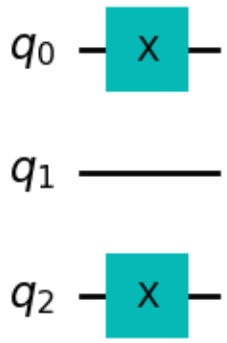
[1]:
```
import numpy as np
from numpy import pi

# importing Qiskit
from qiskit import *
from qiskit import QuantumCircuit, transpile, Aer, IBMQ

# from qiskit.providers.ibmq import least_busy
from qiskit.tools.monitor import job_monitor
from qiskit.visualization import (
    plot_histogram,
    plot_bloch_multivector,
    plot_distribution,
)

from qiskit.quantum_info import Statevector
```

[ ]:
```
sim = Aer.get_backend("aer_simulator")

backend = BasicAer.get_backend("qasm_simulator")
shots = 2048
```

Before performing QFT and IQFT, we will define some helper functions.

[45]:
```
def initialize_circuit(n_qubits: int, number: int) -> QuantumCircuit:
    qc = QuantumCircuit(n_qubits)
    binary_number = format(number, f"0{n_qubits}b")
    for qubit, bit in enumerate(reversed(binary_number)):
        if bit == "1":
            qc.x(qubit)
    return qc
```

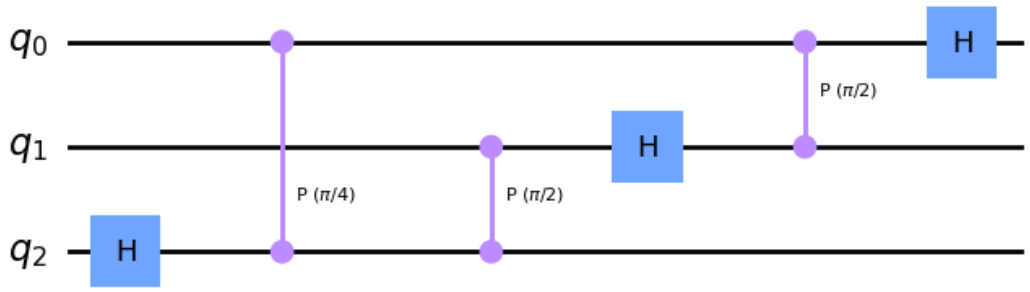
```
# Let's create a 3-qubit circuit and initialize it to the state |5
circuit = initialize_circuit(n_qubits=3, number=5)
display(circuit.draw("mpl"))
```



Below, we define QFT circuit creation. Please note that this is done in a recursive way.

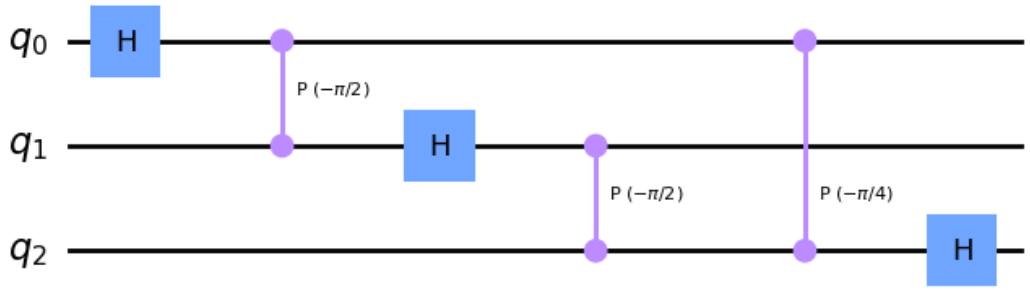
[49]:
```
def qft_rotations(circuit: QuantumCircuit, n: int) -> QuantumCircuit:
    """Performs qft on the first n qubits in circuit (without swaps)"""
    if n == 0:
        return circuit
    n -= 1
    circuit.h(n)
    for qubit in range(n):
        circuit.cp(pi / 2 ** (n - qubit), qubit, n)
    # At the end of our function, we call the same function again on
    # the next qubits (we reduced n by one earlier in the function)
    qft_rotations(circuit, n)


def swap_registers(circuit: QuantumCircuit, n: int) -> QuantumCircuit:
    for qubit in range(n // 2):
        circuit.swap(qubit, n - qubit - 1)
    return circuit


def qft(n: int) -> QuantumCircuit:
    """QFT on the first n qubits in circuit"""
    circuit = QuantumCircuit(n)
    qft_rotations(circuit, n)
```

```
    swap_registers(circuit, n)
    return circuit


n_qubits = 3
qft_circuit = QuantumCircuit(n_qubits)
qft_rotations(qft_circuit, n_qubits)
display(qft_circuit.draw("mpl"))
```



We will create the inverse QFT circuit by taking an inverse of the `qft_circuit`:

[50]:
```
iqft_circuit = qft_circuit.inverse()
display(iqft_circuit.draw("mpl"))
```

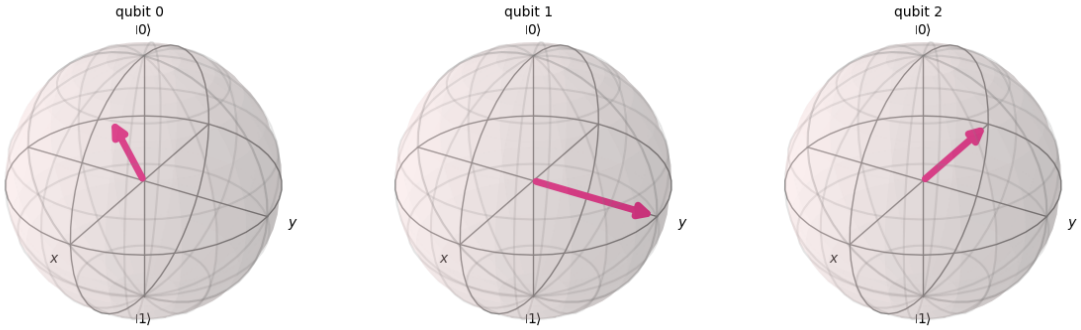


[58]:
```
def iqft_experiment(
    n_qubits: int,
    number: int,
    plot_statevector: bool = False,
    plot_hist: bool = False,
```

```
):

    print(
        f"IQFT of {number} with {n_qubits} qubits. Binary: {format(number,␣
 ↪f'0{n_qubits}b')}."
    )

    initial_circuit = initialize_circuit(n_qubits, number)

    qft_circuit = qft(n_qubits)
    iqft_circuit = qft_circuit.copy().inverse()

    # Combine the initial circuit with the QFT circuit
    qft_circuit = initial_circuit.compose(qft_circuit)

    if plot_statevector:
        # Save the statevector
        qft_copy = qft_circuit.copy()
        qft_copy.save_statevector()
        statevector = sim.run(qft_copy).result().get_statevector()
        display(plot_bloch_multivector(statevector,␣
 ↪title=f"$\widetilde{number}$"))

    # Combine the QFT circuit with the IQFT circuit
    combined_circuit = qft_circuit.compose(iqft_circuit)
    combined_circuit.measure_all()

    transpiled_qc = transpile(combined_circuit, backend, optimization_level=3)
    job = backend.run(transpiled_qc, shots=shots)
    job_monitor(job)

    if plot_statevector:
        transpiled_qc.save_statevector()
        transpiled_qc.measure_all()
        statevector = sim.run(transpiled_qc).result().get_statevector()
        display(plot_bloch_multivector(statevector, title=f"${number}$"))

    if plot_hist:
        counts = job.result().get_counts()
        display(plot_histogram(counts))


iqft_experiment(3, 5, plot_statevector=True, plot_hist=True)
```

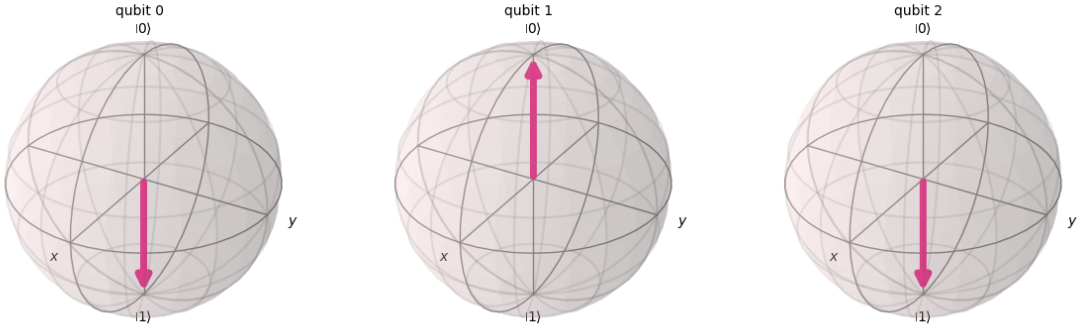
```
IQFT of 5 with 3 qubits. Binary: 101.
```

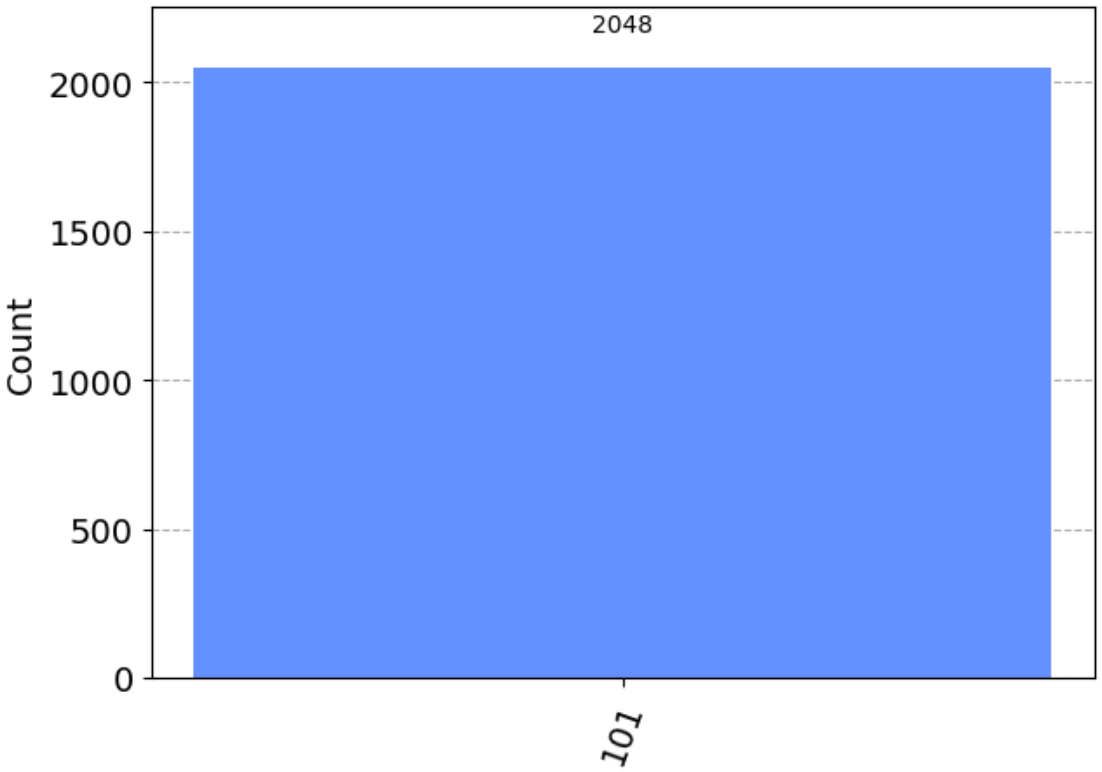
$\tilde{5}$




```
Job Status: job has successfully run
```

5



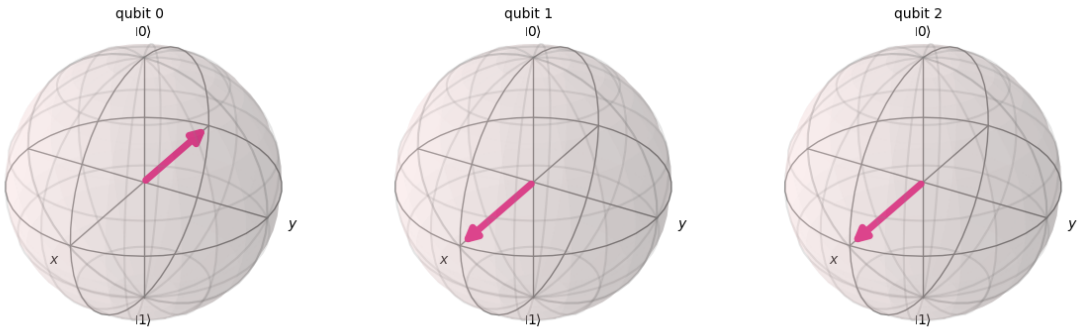


2 Finding a for which QFT is equal to |100>

[54]:
```
iqft_experiment(3, 4, plot_statevector=True, plot_hist=True)
```

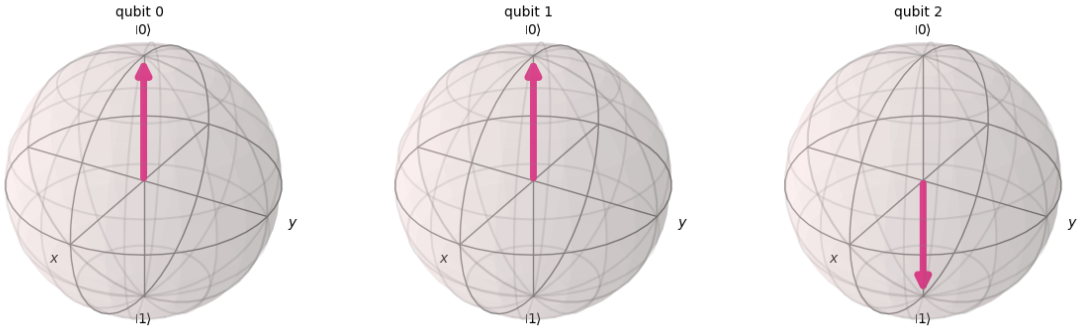
```
IQFT of 4 with 3 qubits. Binary: 100.
```

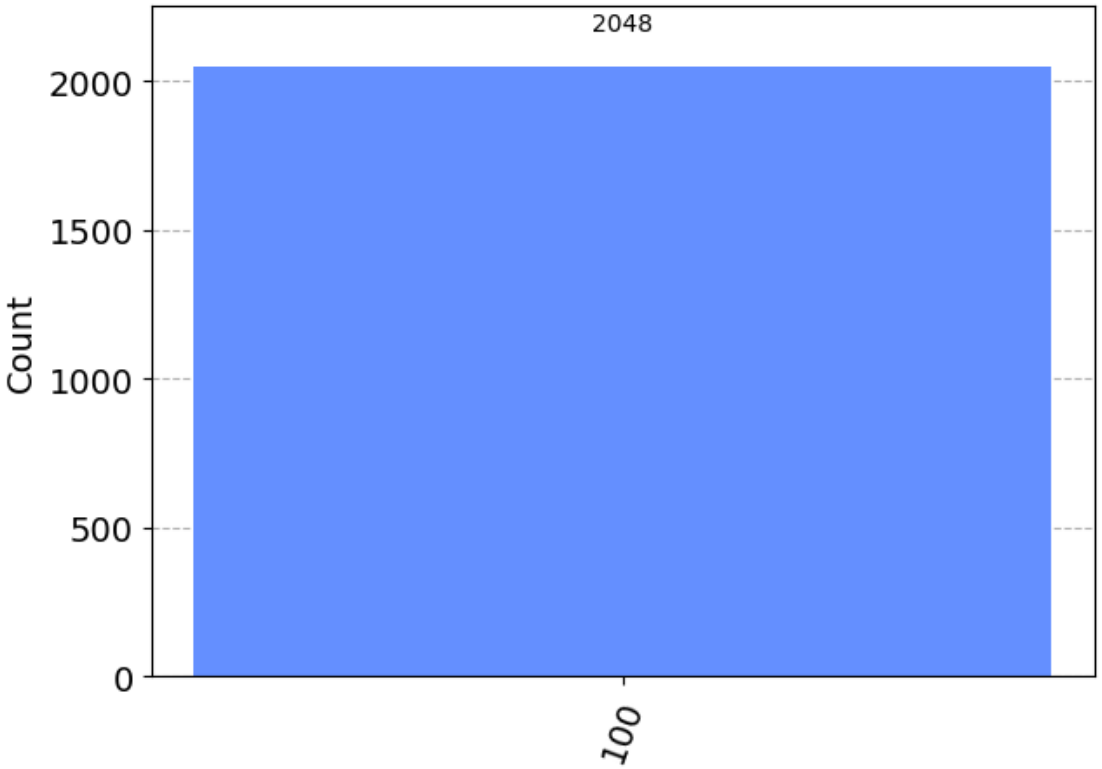
4




```
Job Status: job has successfully run
```

$\tilde{4}$



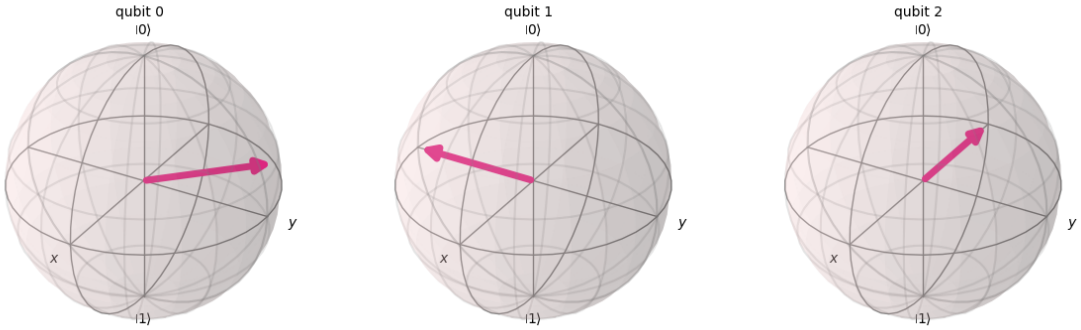


3 Finding b for which QFT is equal to |011>

[56]:
```
iqft_experiment(3, 3, plot_statevector=True, plot_hist=True)
```

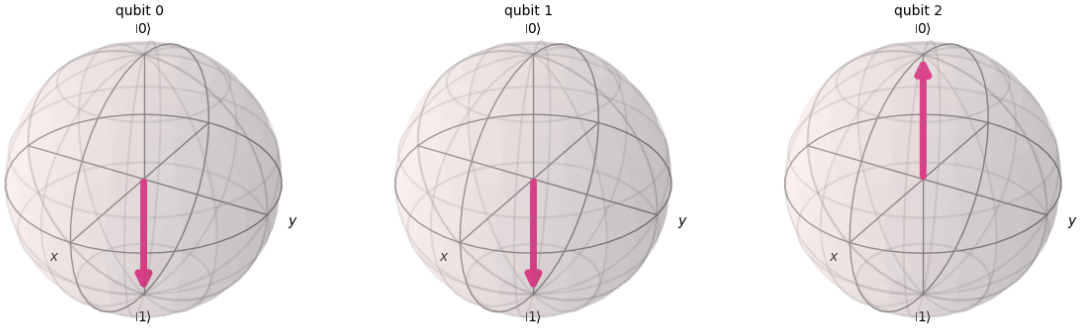
```
IQFT of 3 with 3 qubits. Binary: 011.
```

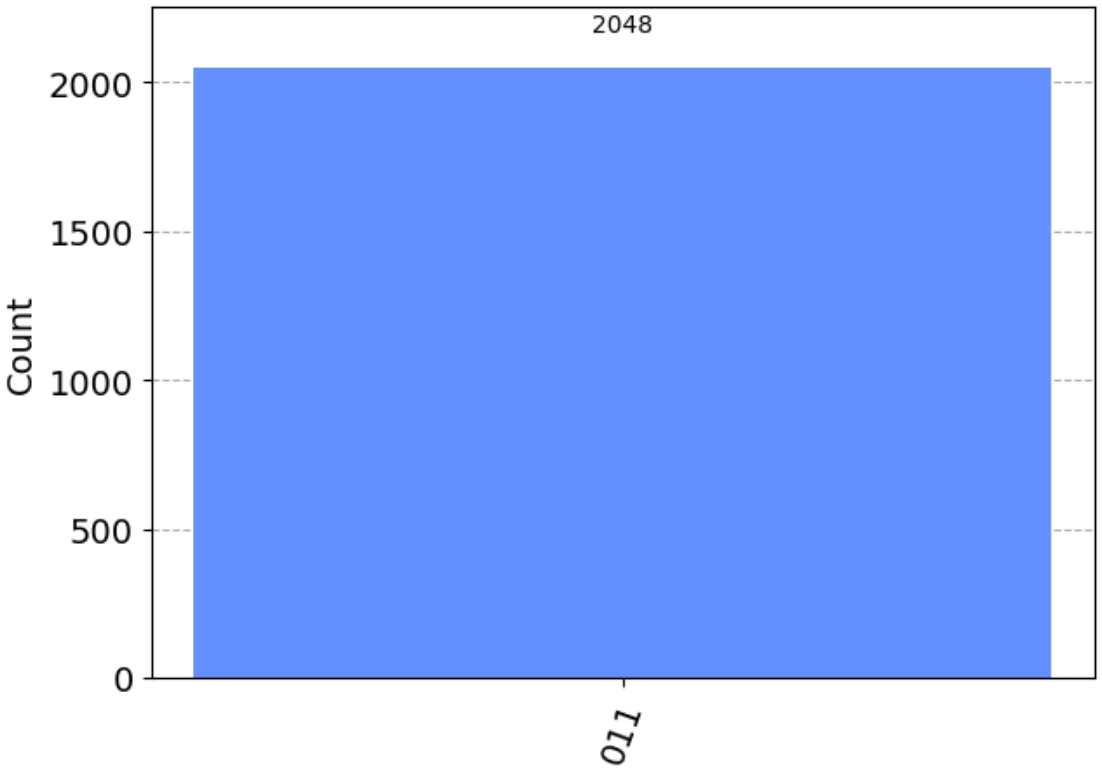
$\tilde{3}$




```
Job Status: job has successfully run
```

3



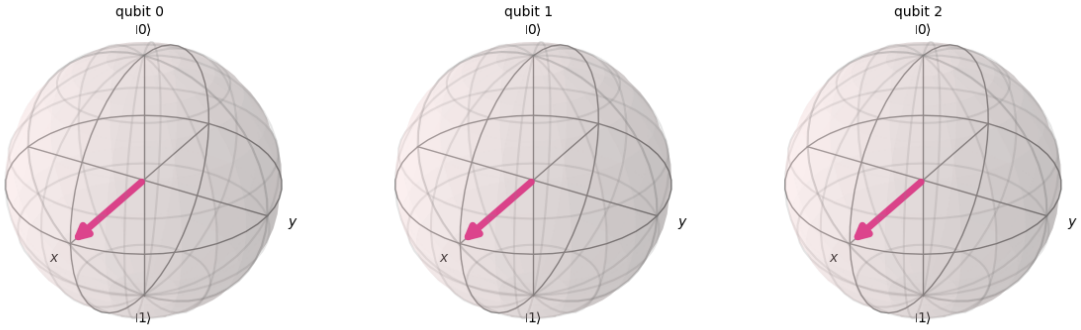


4 Bloch spheres for states from |000> to |111>

[59]:
```
for i in range(8):
    iqft_experiment(3, i, plot_statevector=True, plot_hist=False)
```

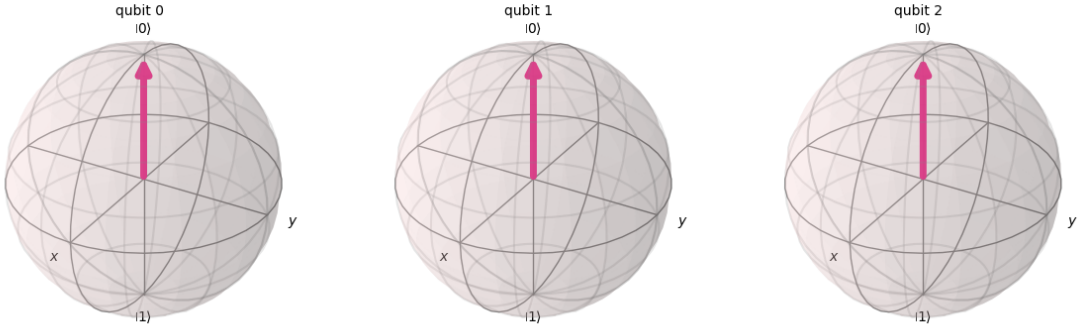
```
IQFT of 0 with 3 qubits. Binary: 000.
```

$\tilde{0}$



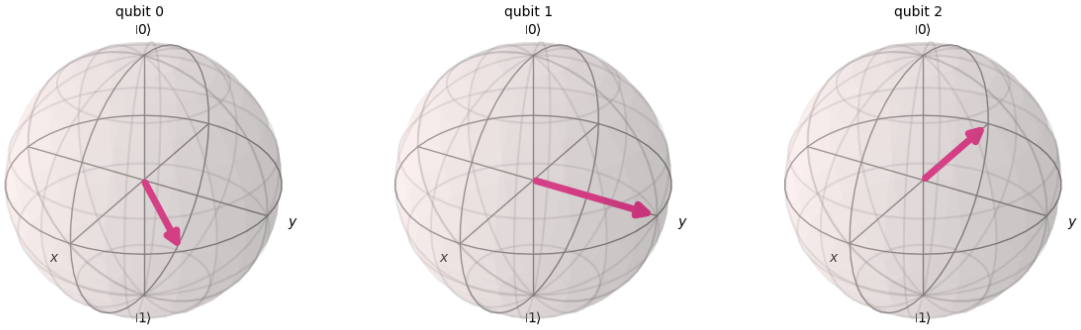
```
Job Status: job has successfully run
```

0



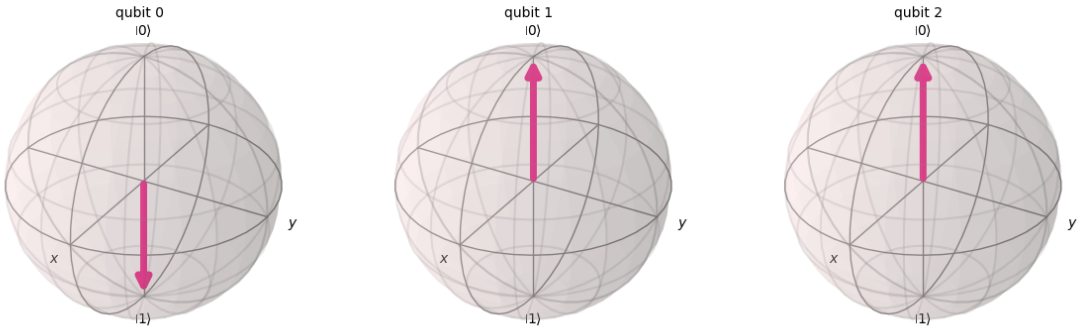

```
IQFT of 1 with 3 qubits. Binary: 001.
```

$\tilde{1}$



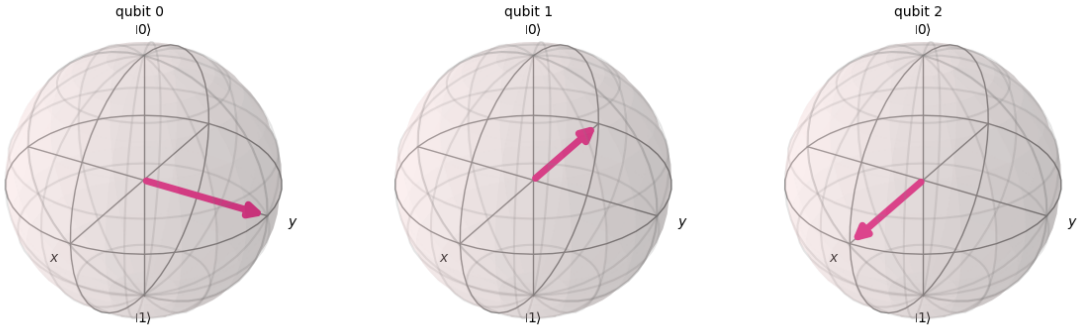

```
Job Status: job has successfully run
```

1



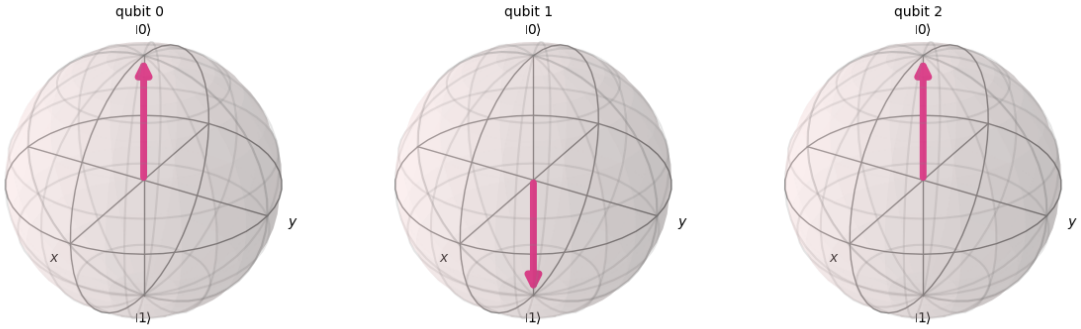
IQFT of 2 with 3 qubits. Binary: 010.

$\tilde{2}$



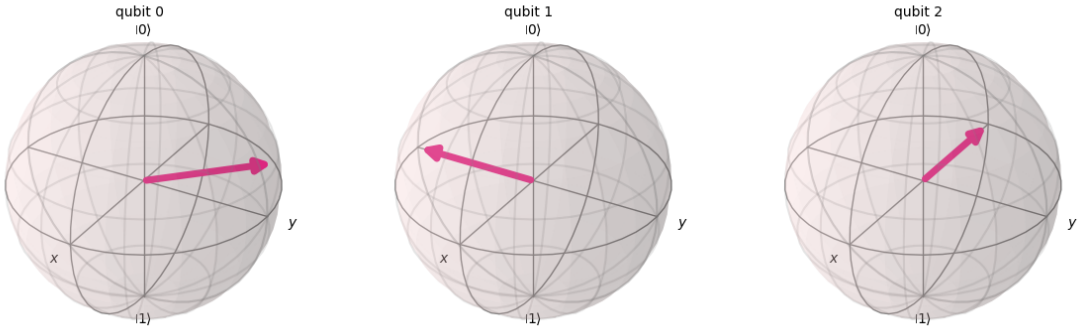
Job Status: job has successfully run

2



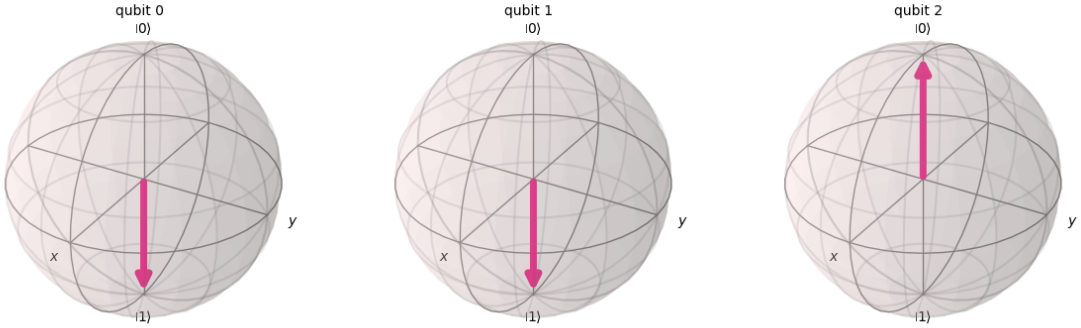
IQFT of 3 with 3 qubits. Binary: 011.

$\tilde{3}$



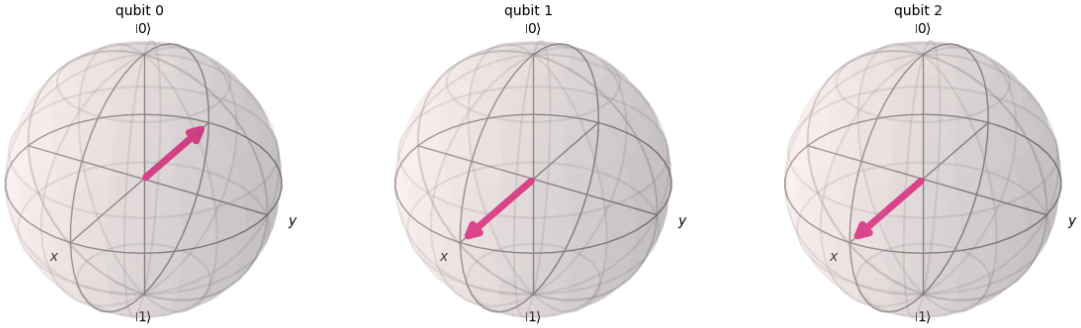

```
Job Status: job has successfully run
```

3



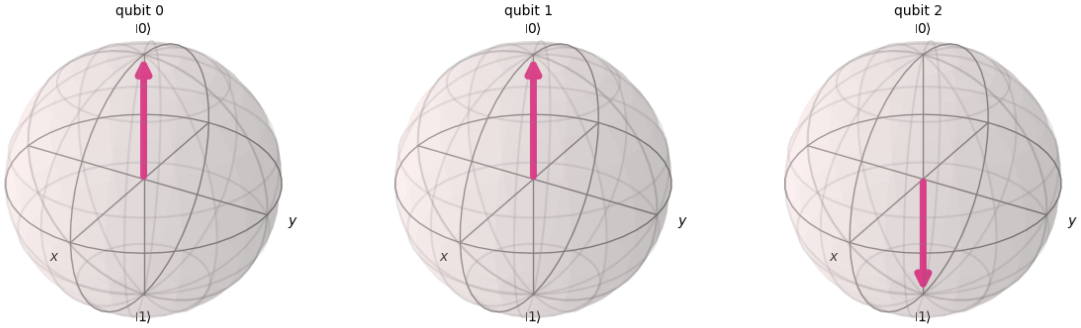

```
IQFT of 4 with 3 qubits. Binary: 100.
```

$\tilde{4}$



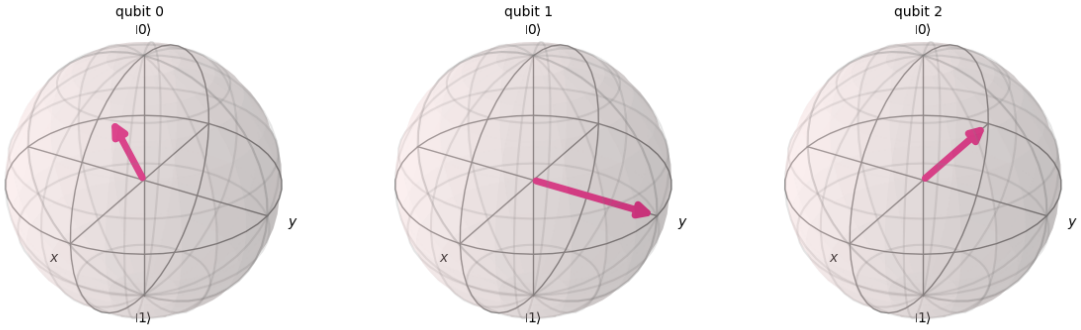
```
Job Status: job has successfully run
```

4



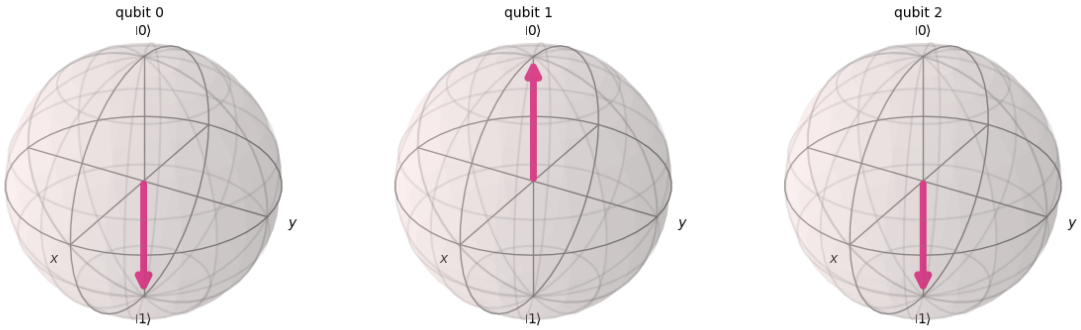

```
IQFT of 5 with 3 qubits. Binary: 101.
```

$\tilde{5}$



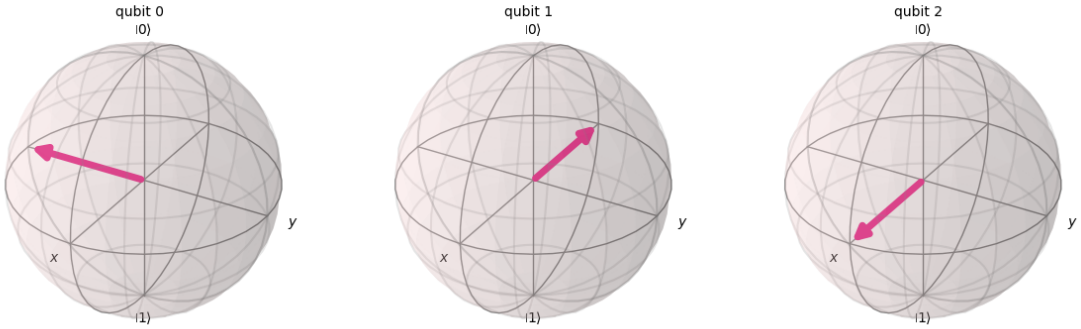

```
Job Status: job has successfully run
```

5



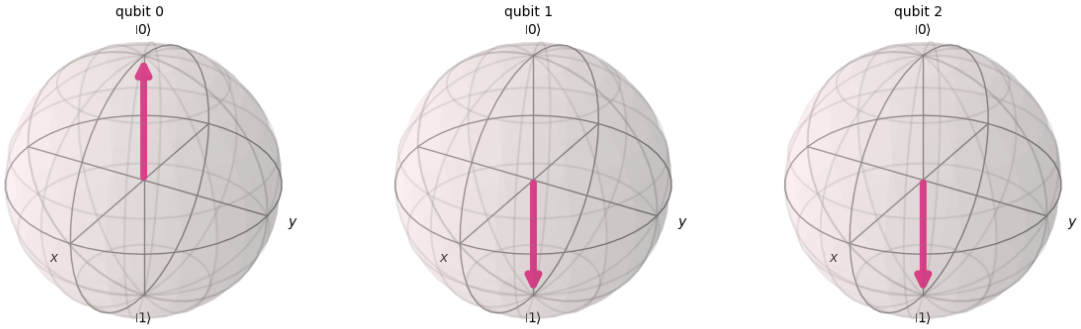
IQFT of 6 with 3 qubits. Binary: 110.

$\tilde{6}$



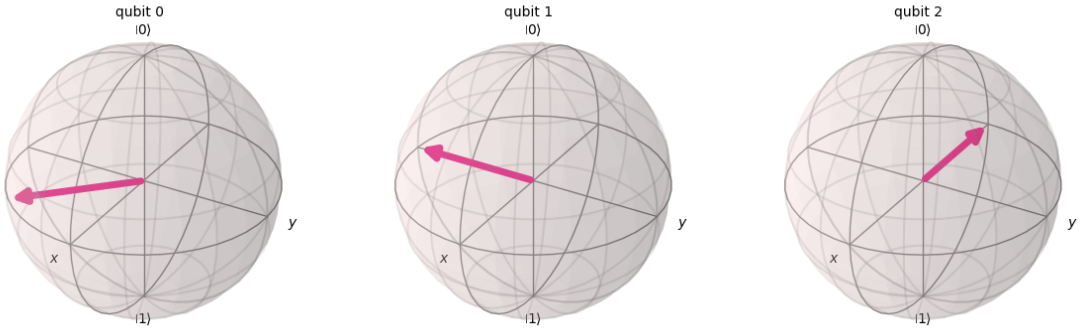
Job Status: job has successfully run

6



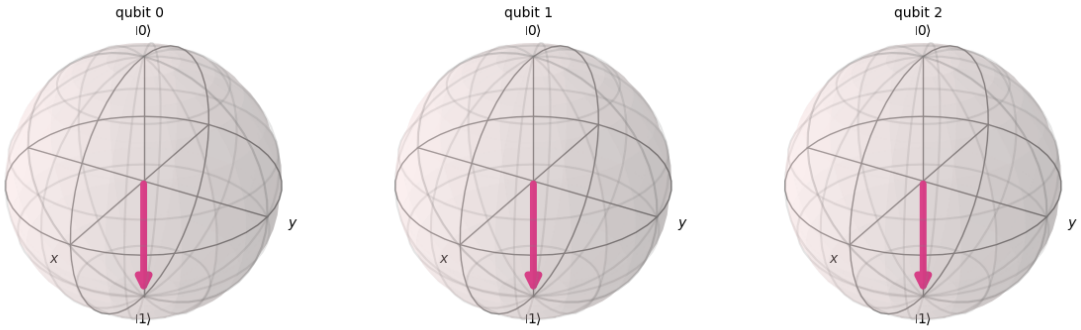
IQFT of 7 with 3 qubits. Binary: 111.

$\tilde{7}$




```
Job Status: job has successfully run
```

7



5 Verification of results with the Unitary Simulator

[62]:
```
from qiskit.execute_function import execute

from qiskit import BasicAer, BasicAerError

backend = BasicAer.get_backend("unitary_simulator")
```

[67]:
```
for i in range(1, 8):
    circuit = initialize_circuit(n_qubits=i, number=1)
    qft_circuit = qft(i)
    combined_circuit = circuit.compose(qft_circuit)

    try:
```

```
        job = execute(combined_circuit, backend)
        result = job.result()
        unitary_mat = result.get_unitary()
    except BasicAerError as ex:
        print("Error:", ex)
```