Project_1

October 15, 2024

1 Introduction to Quantum Information and Quantum Machine Learning

Instructor: Dr Sci. Eng. Przemysław Głowacki

Kacper Dobek 148247

[89]:
```
from qiskit import (
    QuantumRegister,
    ClassicalRegister,
    QuantumCircuit,
    execute,
    Aer,
    BasicAer,
    IBMQ,
)
from qiskit.compiler import transpile, assemble
from qiskit.visualization import (
    plot_state_city,
    plot_bloch_multivector,
    plot_state_hinton,
    plot_state_qsphere,
    plot_histogram,
    plot_distribution,
)
from numpy import pi
import matplotlib.pyplot as plt
```

[90]:
```
N_RUNS = 2048
```

[91]:
```
# selection of quantum simulator (or processor)
QASM_BACKEND = Aer.get_backend("qasm_simulator")
STATEVECTOR_BACKEND = BasicAer.get_backend("statevector_simulator")
```

Let's define a class for a quantum experiment to make execution more elegant.

[92]:
```
class QuantumExperiment:
    def __init__(self, get_circuit: callable, name: str = "Experiment"):
        self.circuit = get_circuit()
```

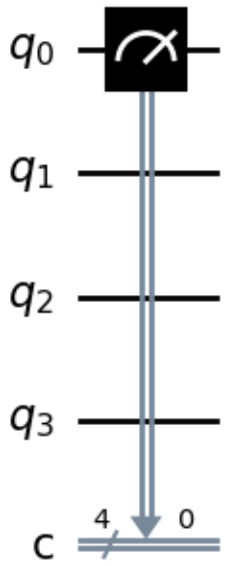
```
        self.name = name

    def run(self, shots: int = N_RUNS):
        print(f"Running experiment: {self.name}")

        print("The circuit")
        display(self.circuit.draw(output="mpl"))
        self.run_qasm(shots)
        self.run_statevector()

    def run_qasm(self, shots: int, n_reps: int = 3):
        # Execute the circuit on the qasm simulator
        results = []
        for _ in range(n_reps):
            job_sim = execute(self.circuit, QASM_BACKEND, shots=shots)
            results.append(job_sim.result().get_counts(self.circuit))
        # Print the result
        print(f"Execution result (over {n_reps} repetitions): ", results)
        legend = [f"Execution {i}" for i in range(n_reps)]
        # Plot the histogram
        print("Histogram")
        display(
            plot_histogram(results, title=f"Histogram for {self.name}",␣
↪legend=legend)
        )

        # Plot probability distribution
        print("Probability distribution")
        display(
            plot_distribution(
                results,
                title=f"Probability distribution for {self.name}",
                legend=legend,
            )
        )

    def run_statevector(self):
        # Execute the circuit on the statevector simulator
        result = STATEVECTOR_BACKEND.run(
            transpile(self.circuit, STATEVECTOR_BACKEND)
        ).result()
        psi = result.get_statevector(self.circuit)
        # Plot state city
        print("State city")
        display(plot_state_city(psi, title=f"State city for {self.name}"))

        # Plot Q-sphere
```

```
        print("Q-sphere")
        display(plot_state_qsphere(psi, figsize=(5, 5)))
```

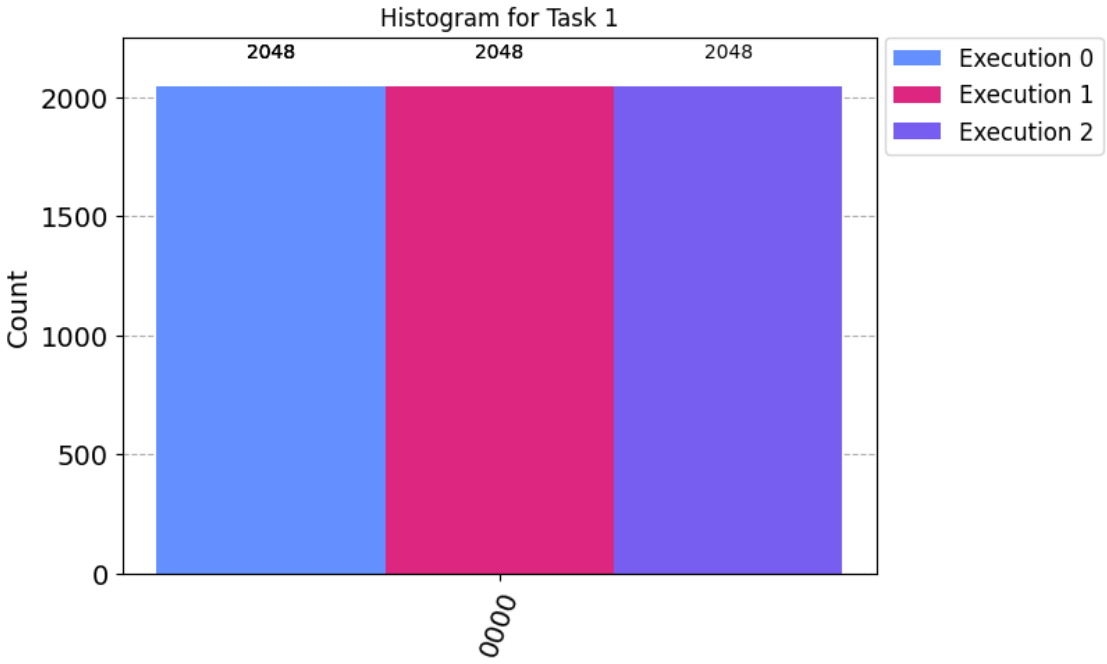
1.0.1 Task 1

[93]:
```
def task1_circuit():
    qreg_q = QuantumRegister(4, "q")
    creg_c = ClassicalRegister(4, "c")
    circuit = QuantumCircuit(qreg_q, creg_c)

    circuit.measure(qreg_q[0], creg_c[0])

    return circuit


quantum_experiment_1 = QuantumExperiment(task1_circuit, name="Task 1")
quantum_experiment_1.run()
```

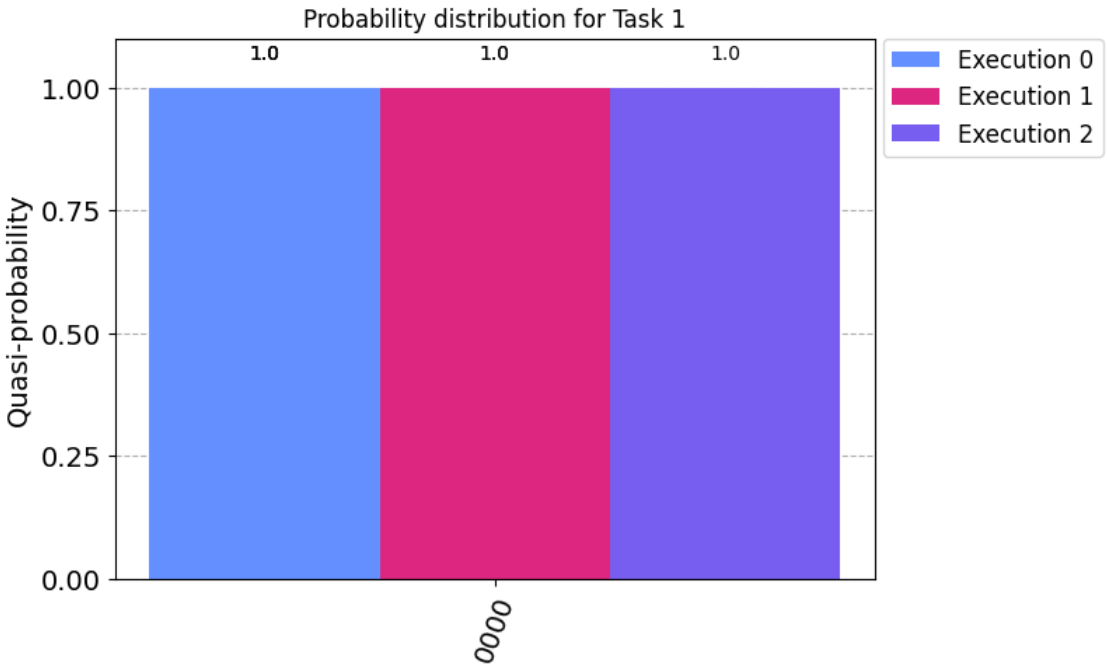
```
Running experiment: Task 1
The circuit
```




```
Execution result (over 3 repetitions):  [{'0000': 2048}, {'0000': 2048},
{'0000': 2048}]
Histogram
```

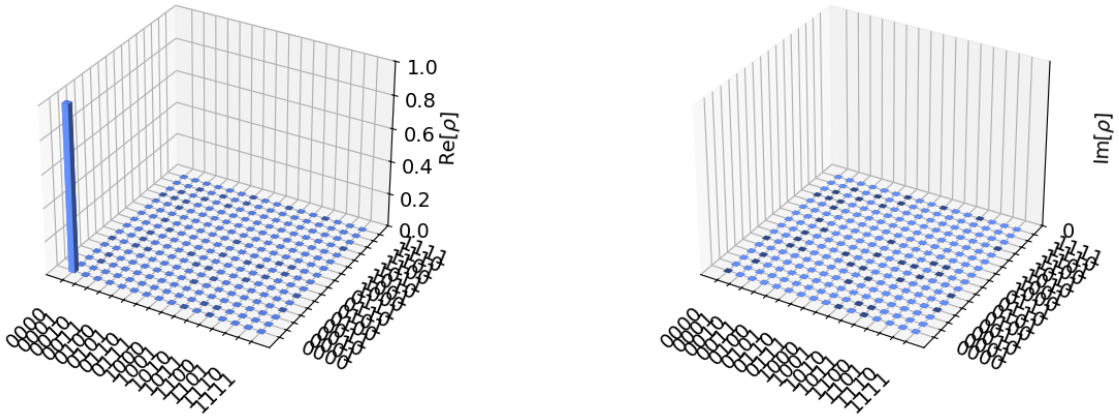


Probability distribution

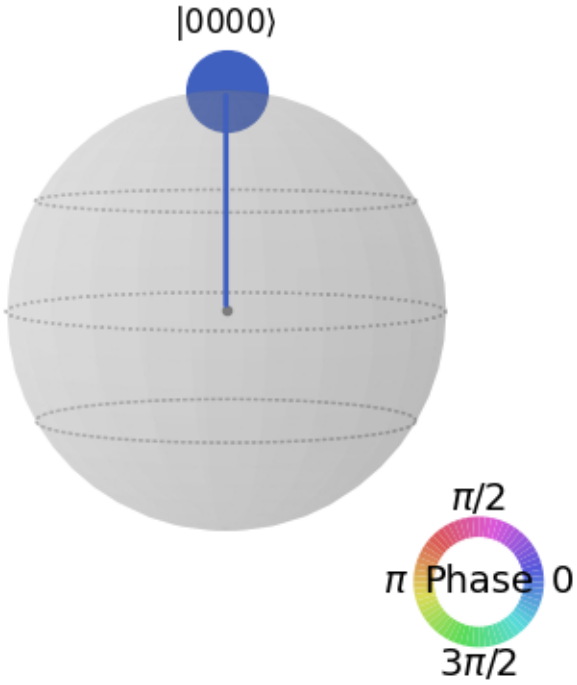


State city

State city for Task 1



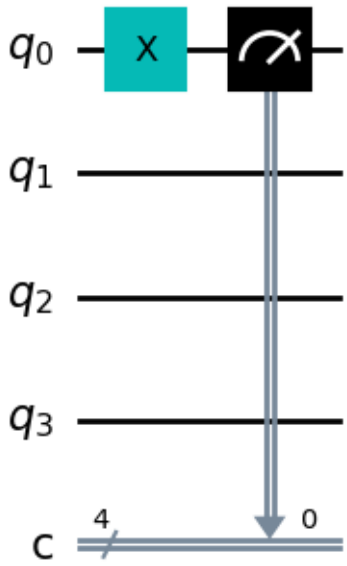
Q-sphere



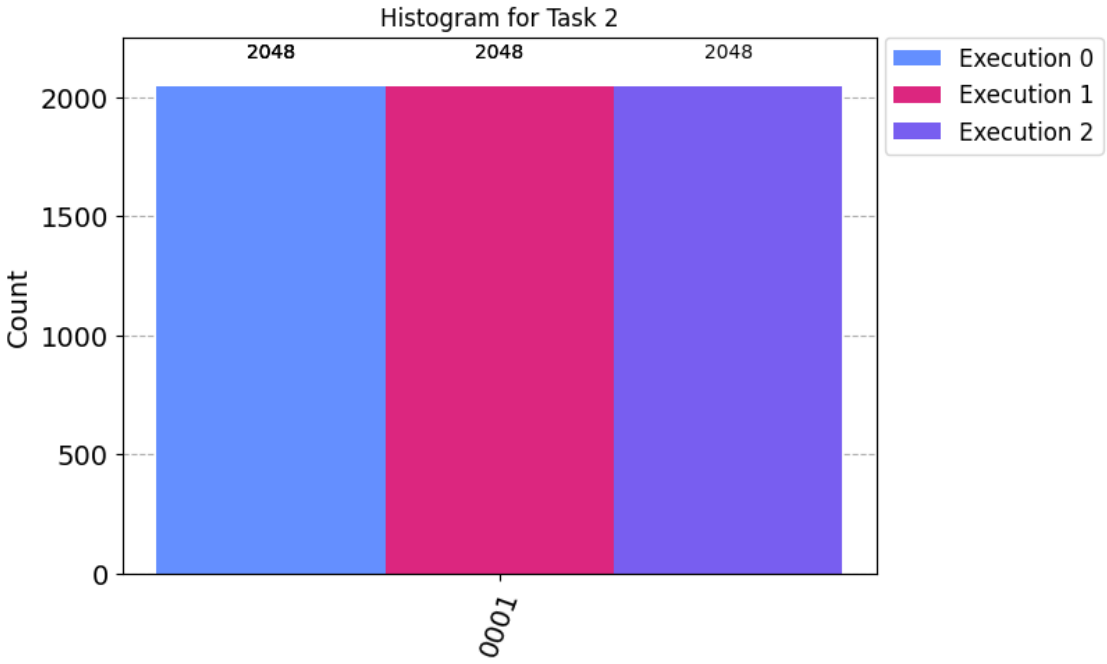
1.0.2 Task 2

[94]:
```
def task2_circuit():
    qreg_q = QuantumRegister(4, "q")
    creg_c = ClassicalRegister(4, "c")
    circuit = QuantumCircuit(qreg_q, creg_c)

    circuit.x(qreg_q[0])
    circuit.measure(qreg_q[0], creg_c[0])

    return circuit


quantum_experiment_2 = QuantumExperiment(task2_circuit, name="Task 2")
quantum_experiment_2.run()
```

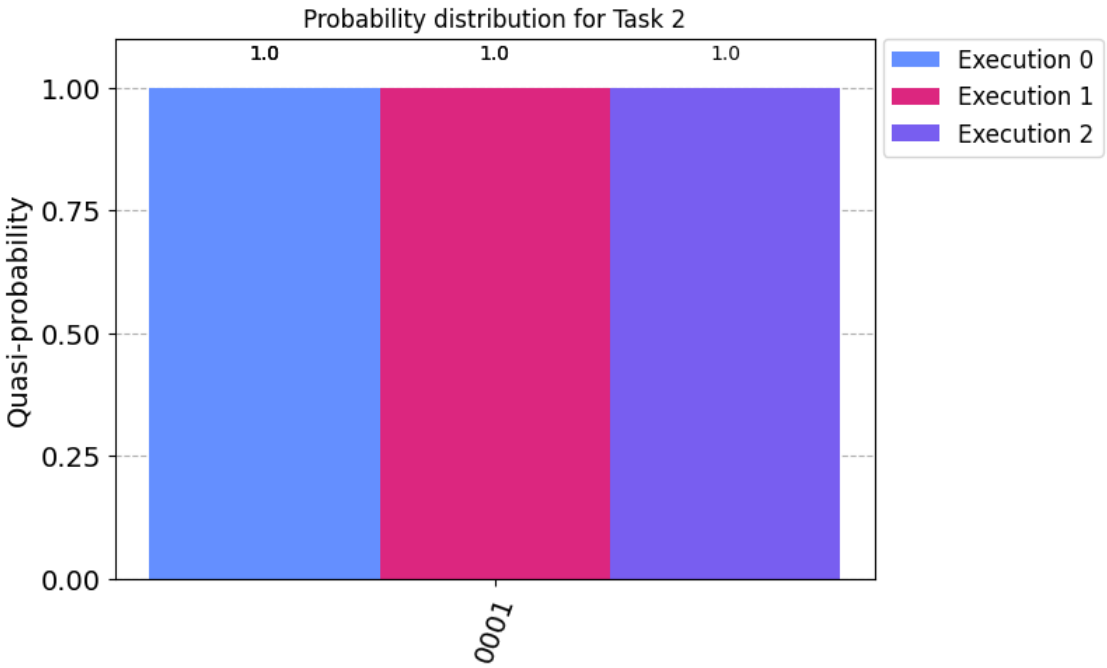
```
Running experiment: Task 2
The circuit
```




```
Execution result (over 3 repetitions):  [{'0001': 2048}, {'0001': 2048},
{'0001': 2048}]
Histogram
```

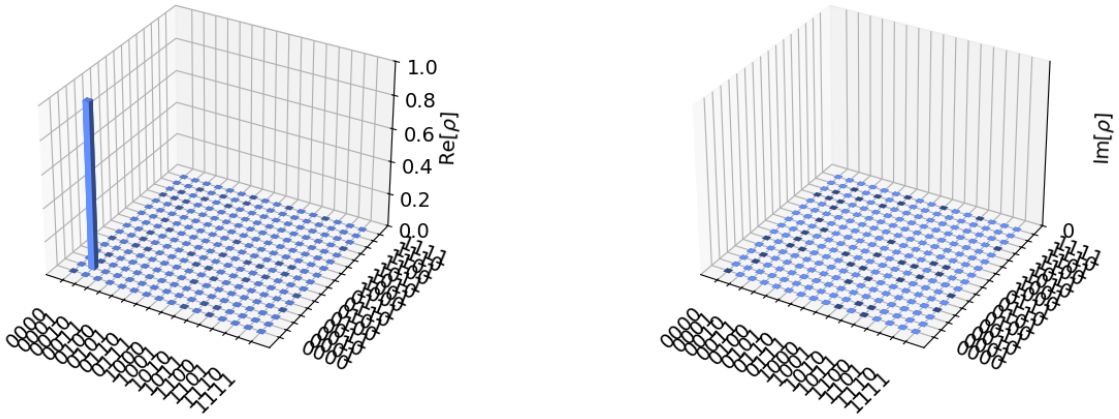


Probability distribution

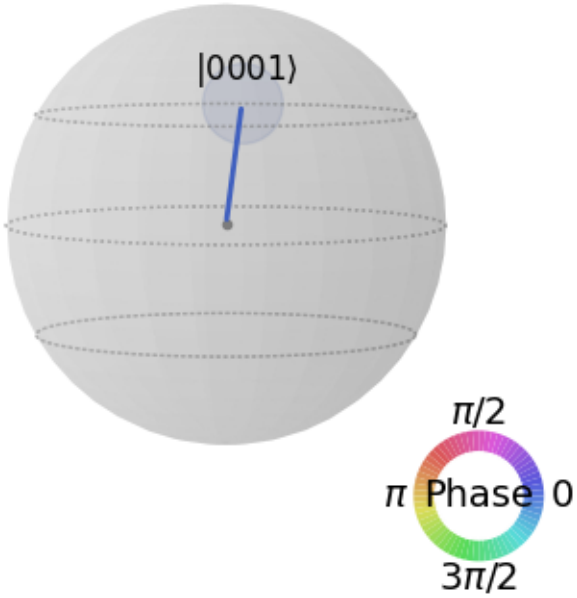


State city

State city for Task 2



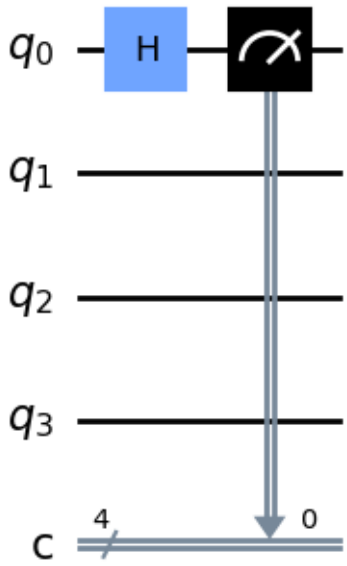
Q-sphere



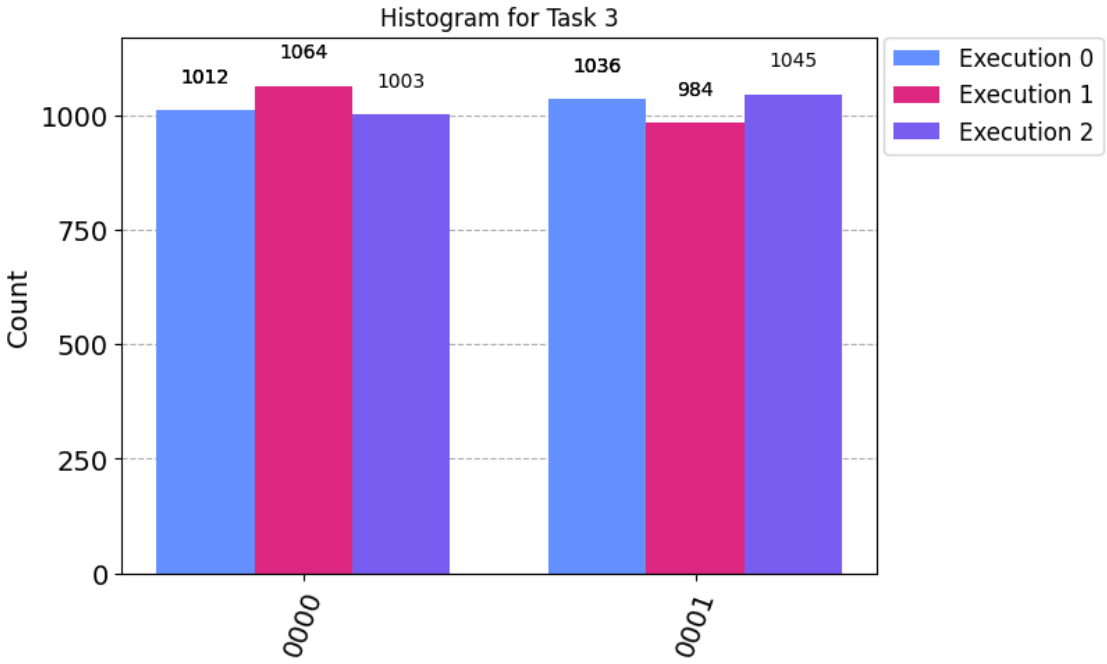
1.0.3 Task 3

[95]:
```
def task3_circuit():
    qreg_q = QuantumRegister(4, "q")
    creg_c = ClassicalRegister(4, "c")
    circuit = QuantumCircuit(qreg_q, creg_c)

    circuit.h(qreg_q[0])
    circuit.measure(qreg_q[0], creg_c[0])

    return circuit


quantum_experiment_3 = QuantumExperiment(task3_circuit, name="Task 3")
quantum_experiment_3.run()
```

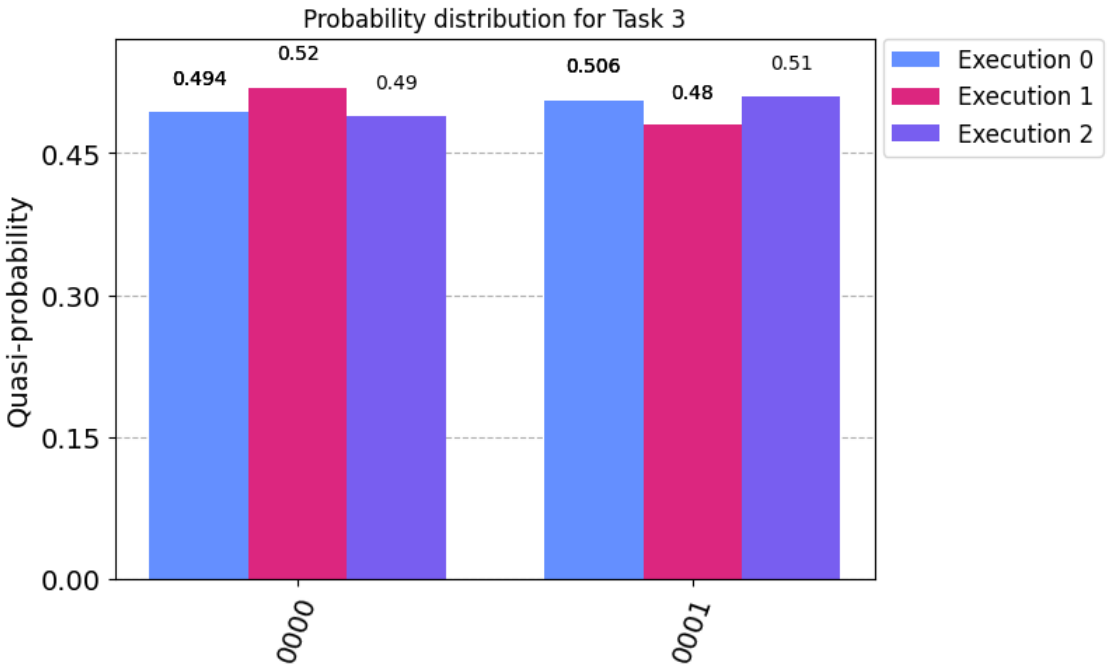
```
Running experiment: Task 3
The circuit
```




```
Execution result (over 3 repetitions):  [{'0001': 1036, '0000': 1012}, {'0000':
1064, '0001': 984}, {'0001': 1045, '0000': 1003}]
Histogram
```

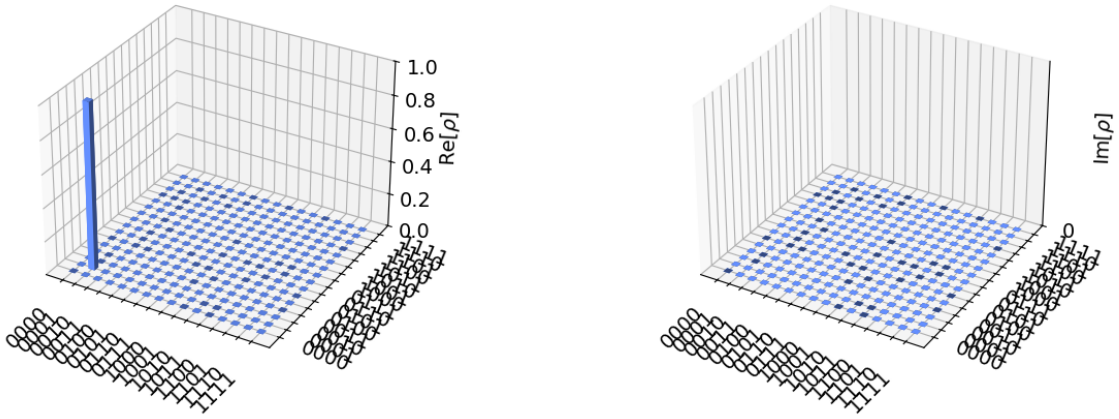


Probability distribution

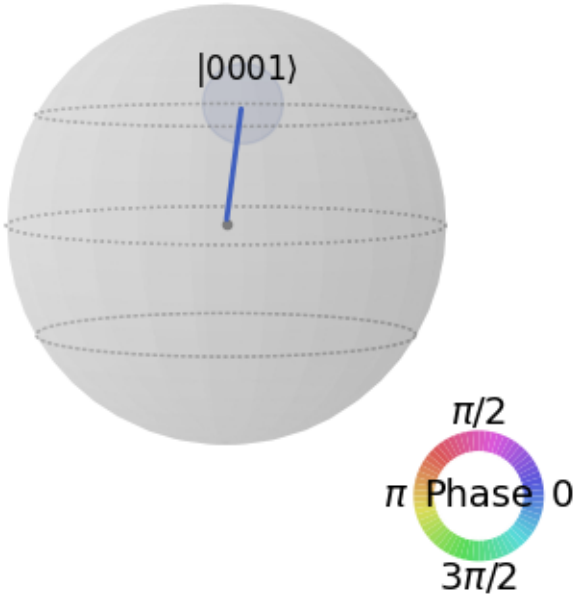


State city

State city for Task 3



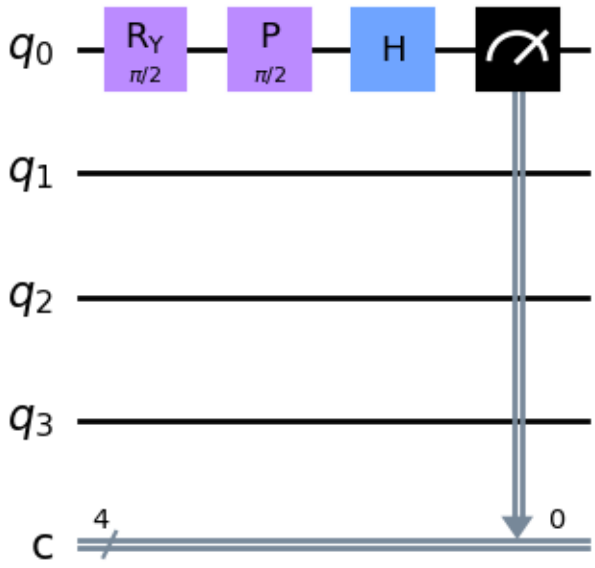
Q-sphere



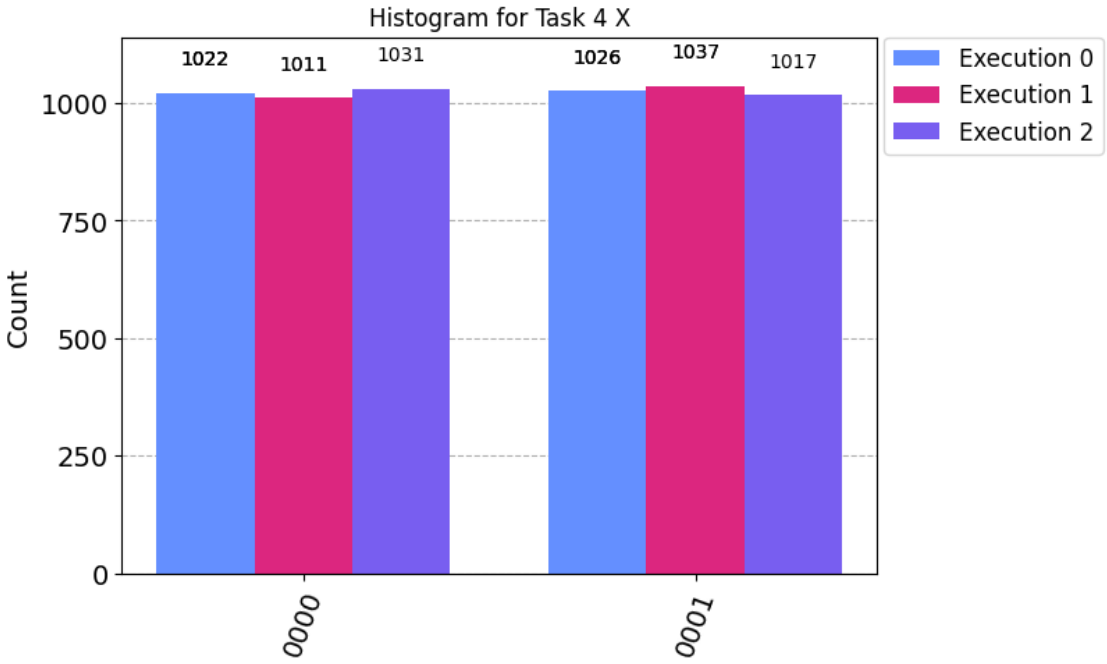
1.0.4 Task 4

[96]:
```
def task4x_circuit():
    qreg_q = QuantumRegister(4, "q")
    creg_c = ClassicalRegister(4, "c")
    circuit = QuantumCircuit(qreg_q, creg_c)

    circuit.ry(pi / 2, qreg_q[0])
    circuit.p(pi / 2, qreg_q[0])
    circuit.h(qreg_q[0])
    circuit.measure(qreg_q[0], creg_c[0])

    return circuit


quantum_experiment_4 = QuantumExperiment(task4x_circuit, name="Task 4 X")
quantum_experiment_4.run()
```

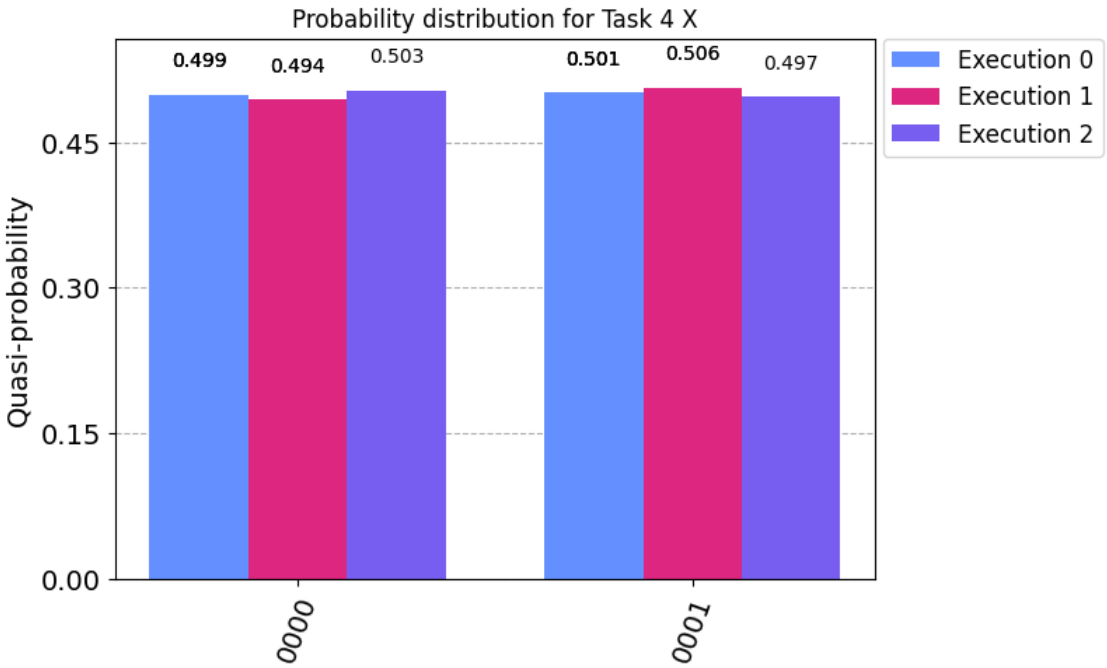
```
Running experiment: Task 4 X
The circuit
```




```
Execution result (over 3 repetitions):  [{'0000': 1022, '0001': 1026}, {'0001':
1037, '0000': 1011}, {'0000': 1031, '0001': 1017}]
Histogram
```

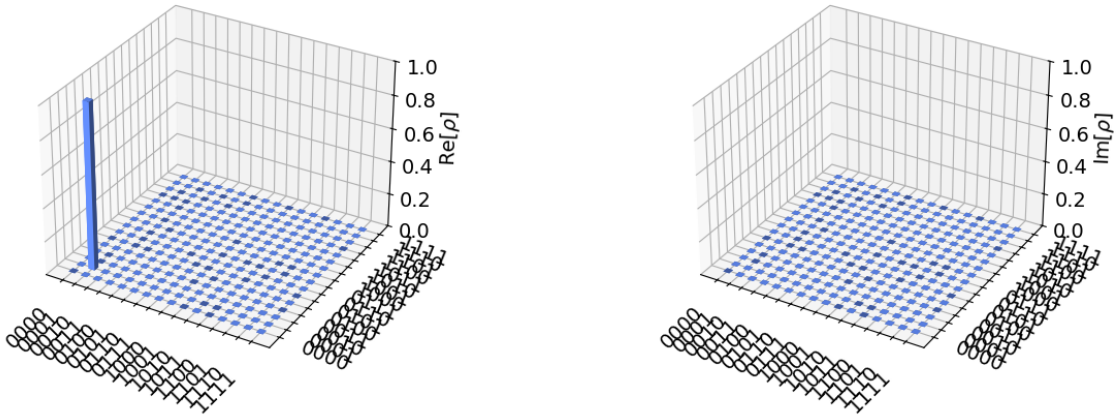


Probability distribution

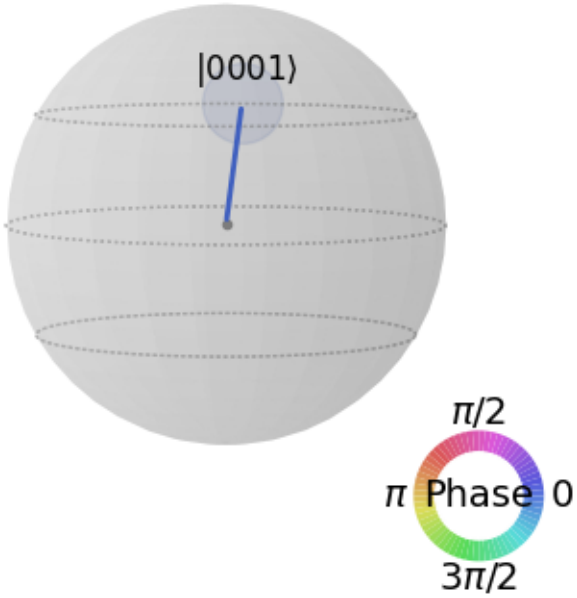


State city

State city for Task 4 X



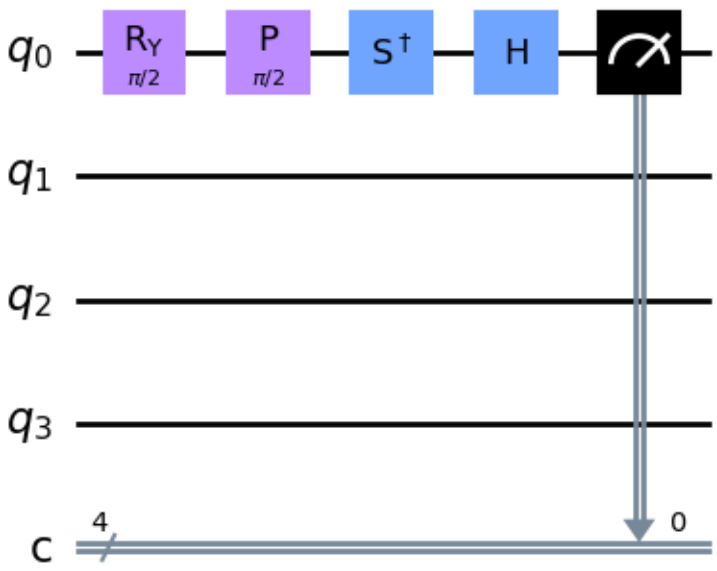
Q-sphere



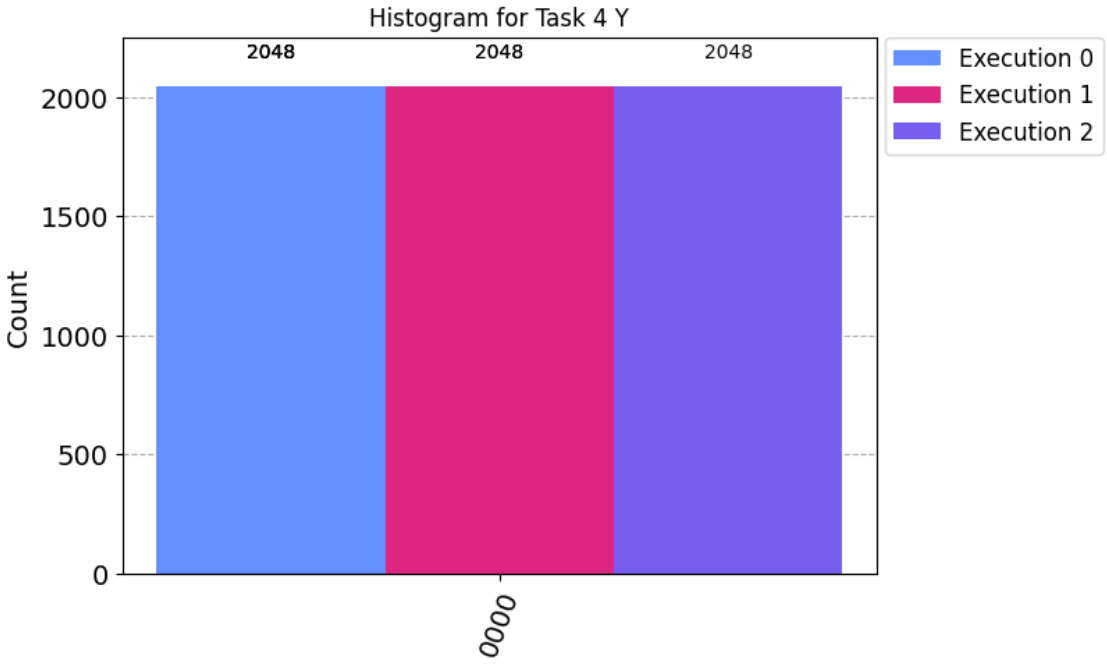
[97]:
```
def task4y_circuit():
    qreg_q = QuantumRegister(4, "q")
    creg_c = ClassicalRegister(4, "c")
```

```
    circuit = QuantumCircuit(qreg_q, creg_c)

    circuit.ry(pi / 2, qreg_q[0])
    circuit.p(pi / 2, qreg_q[0])
    circuit.sdg(qreg_q[0])
    circuit.h(qreg_q[0])
    circuit.measure(qreg_q[0], creg_c[0])

    return circuit


quantum_experiment_4 = QuantumExperiment(task4y_circuit, name="Task 4 Y")
quantum_experiment_4.run()
```

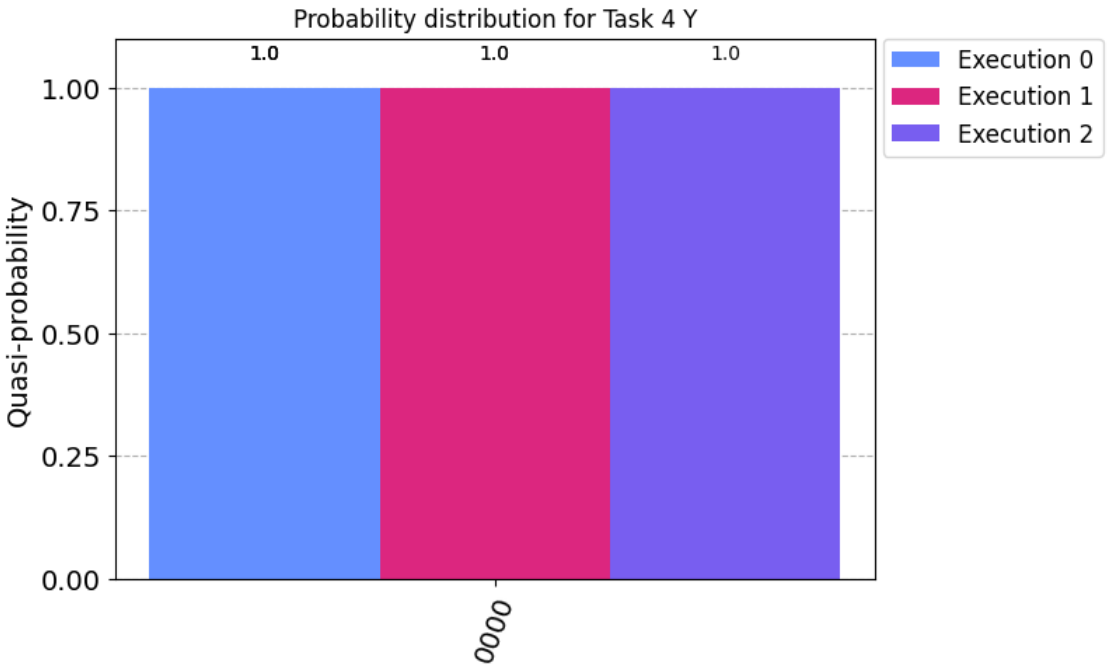
```
Running experiment: Task 4 Y
The circuit
```




```
Execution result (over 3 repetitions):  [{'0000': 2048}, {'0000': 2048},
{'0000': 2048}]
Histogram
```

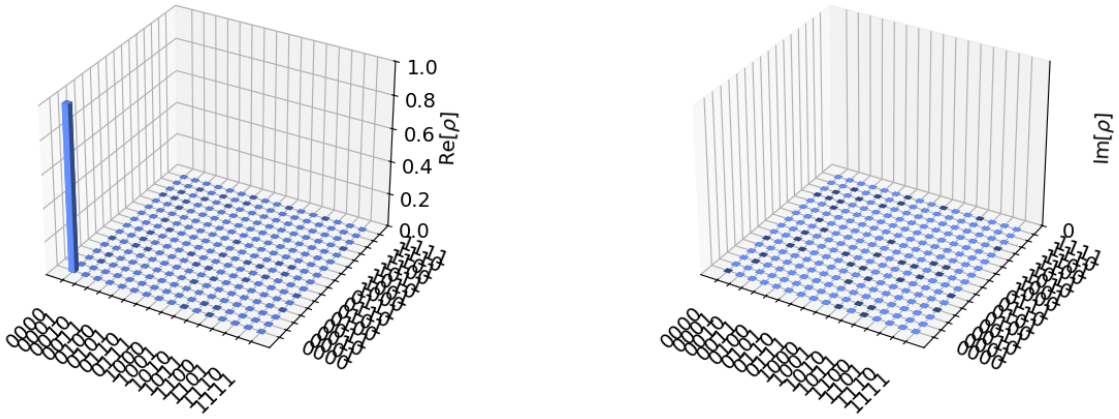


Probability distribution

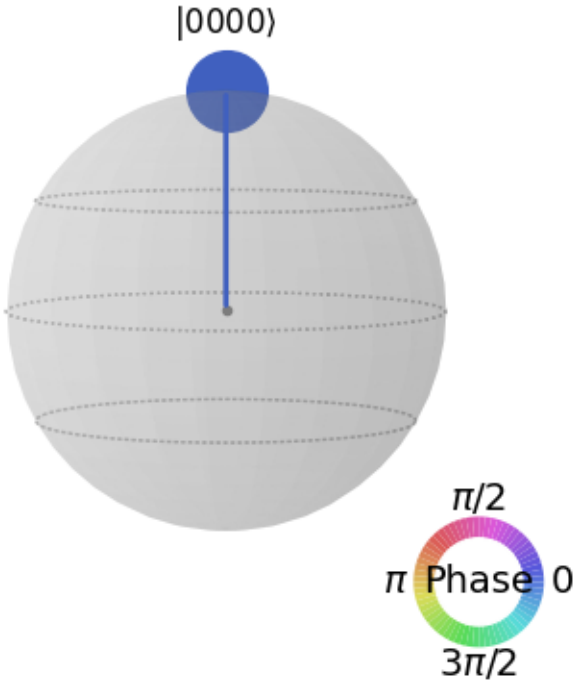


State city

State city for Task 4 Y



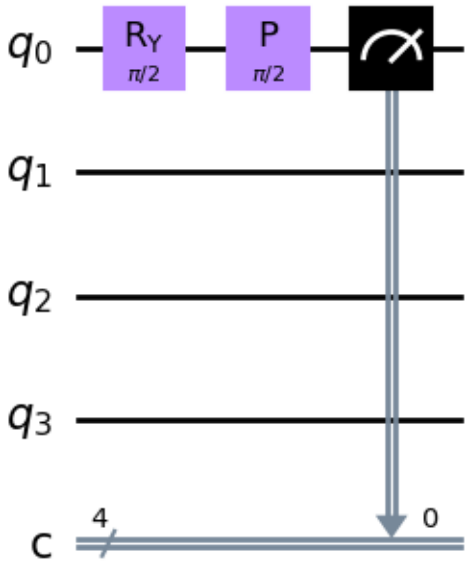
Q-sphere



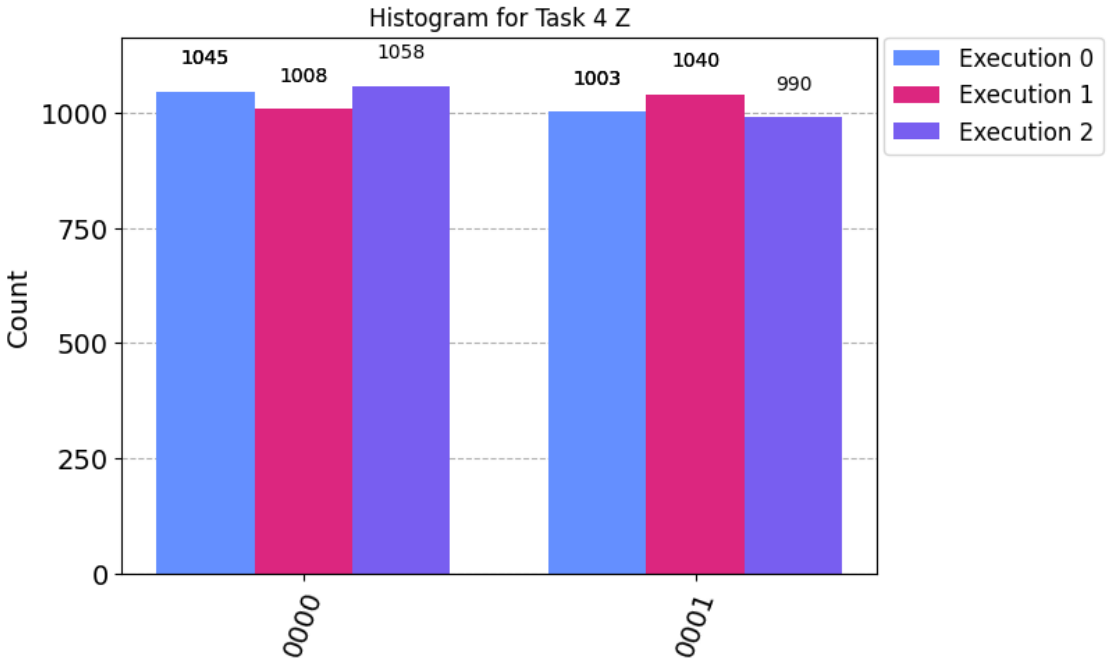
[98]:
```
def task4z_circuit():
    qreg_q = QuantumRegister(4, "q")
    creg_c = ClassicalRegister(4, "c")
```

```
    circuit = QuantumCircuit(qreg_q, creg_c)

    circuit.ry(pi / 2, qreg_q[0])
    circuit.p(pi / 2, qreg_q[0])
    circuit.measure(qreg_q[0], creg_c[0])

    return circuit


quantum_experiment_4 = QuantumExperiment(task4z_circuit, name="Task 4 Z")
quantum_experiment_4.run()
```

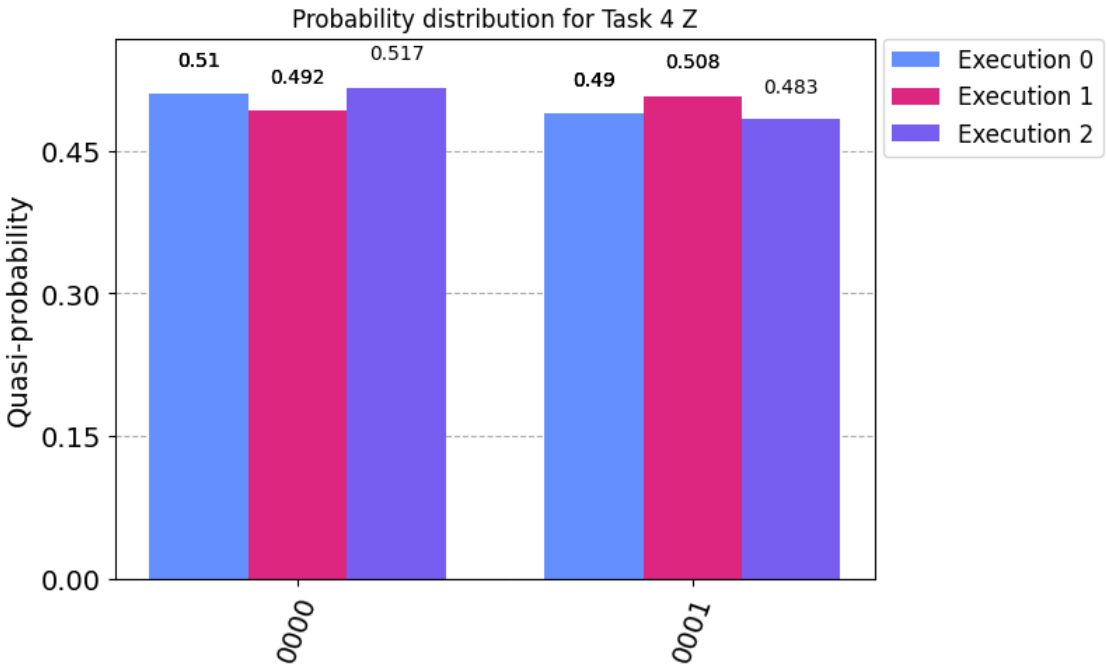
```
Running experiment: Task 4 Z
The circuit
```




```
Execution result (over 3 repetitions):  [{'0000': 1045, '0001': 1003}, {'0001':
1040, '0000': 1008}, {'0001': 990, '0000': 1058}]
Histogram
```

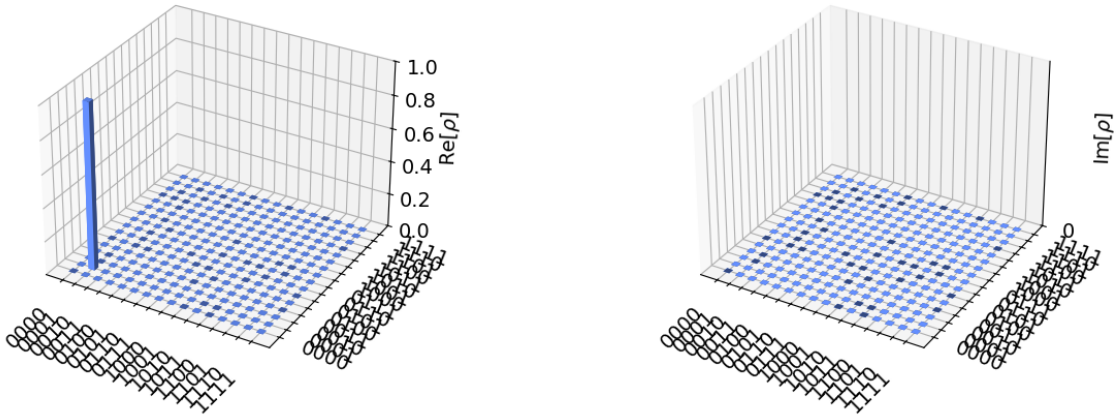


Probability distribution



State city

State city for Task 4 Z



Q-sphere