Michał Kałmucki 151944

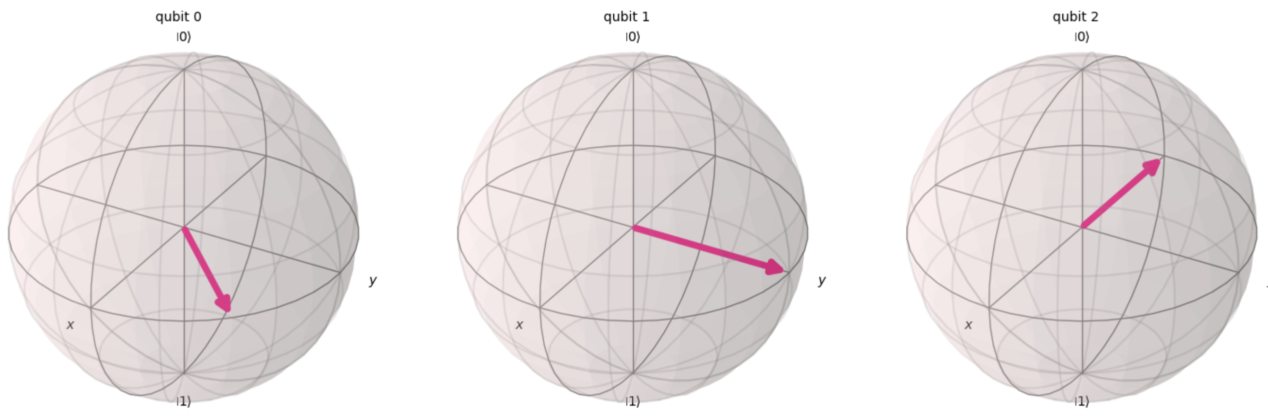
QFT Quantum Fourier Transform

Helper functions

```
def encode_big_endian(binary_string):
    n = len(binary_string)
    qc = QuantumCircuit(n)
    for i, bit in enumerate(binary_string):
        if bit == '1':
            qc.x(i)  # MSB → qubit 0
    return qc

def qft_big_endian(qc, qubits, inverse=False):
    n = len(qubits)
    if not inverse:
        for j in reversed(range(n)):
            qc.h(qubits[j])
            for k in reversed(range(j)):
                qc.cp(np.pi / 2**(j - k), qubits[k], qubits[j])
    else:  # Inverse QFT
        for j in range(n):
            for k in range(j):
                qc.cp(-np.pi / 2**(j - k), qubits[k], qubits[j])
            qc.h(qubits[j])
    # Swap qubits to finalize QFT
    for i in range(n // 2):
        qc.swap(qubits[i], qubits[n - 1 - i])
    return qc


def recursive_qft_big_endian(qc, qubits):
    if len(qubits) == 0:
        return
    target = qubits[-1]
    qc.h(target)
    for i, ctrl in enumerate(qubits[:-1]):
        qc.cp(np.pi / 2**(len(qubits)-1-i), ctrl, target)
    recursive_qft_big_endian(qc, qubits[:-1])

def build_recursive_qft(n_qubits):
    qc = QuantumCircuit(n_qubits)
    qubits = list(range(n_qubits))
    recursive_qft_big_endian(qc, qubits)
    for i in range(n_qubits // 2):
        qc.swap(i, n_qubits - 1 - i)
    return qc

def qft_inverse_statevector(binary_string):
    n = len(binary_string)
    qc = encode_big_endian(binary_string)
    qft_big_endian(qc, list(range(n)))
    qc.save_statevector()
    return sim.run(qc).result().get_statevector()
```

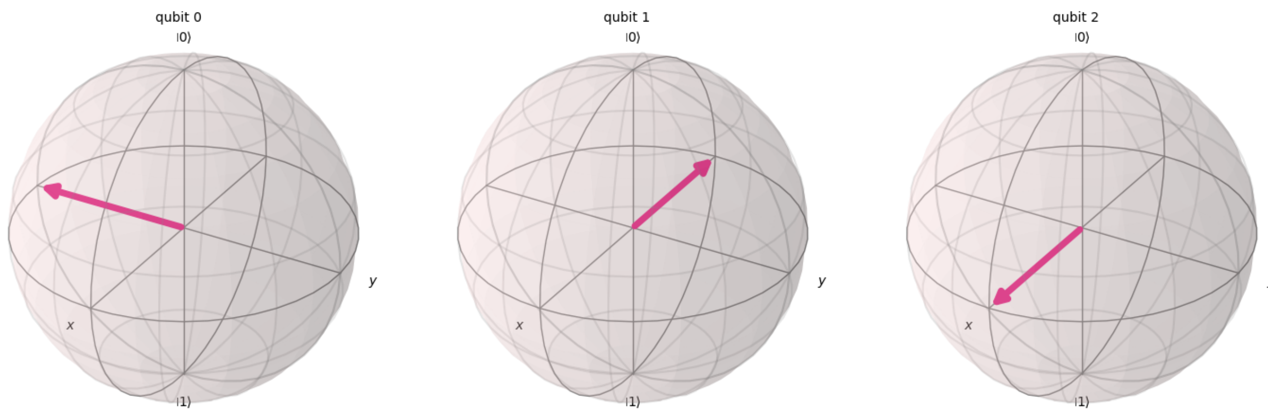
Task 1: Find $|a\rangle$ such that $QFT^\dagger|a\rangle = |100\rangle$

```
state_a = qft_inverse_statevector("100")
plot_bloch_multivector(state_a)
```



Task 2: Find $|b\rangle$ such that $QFT^\dagger|b\rangle = |011\rangle$

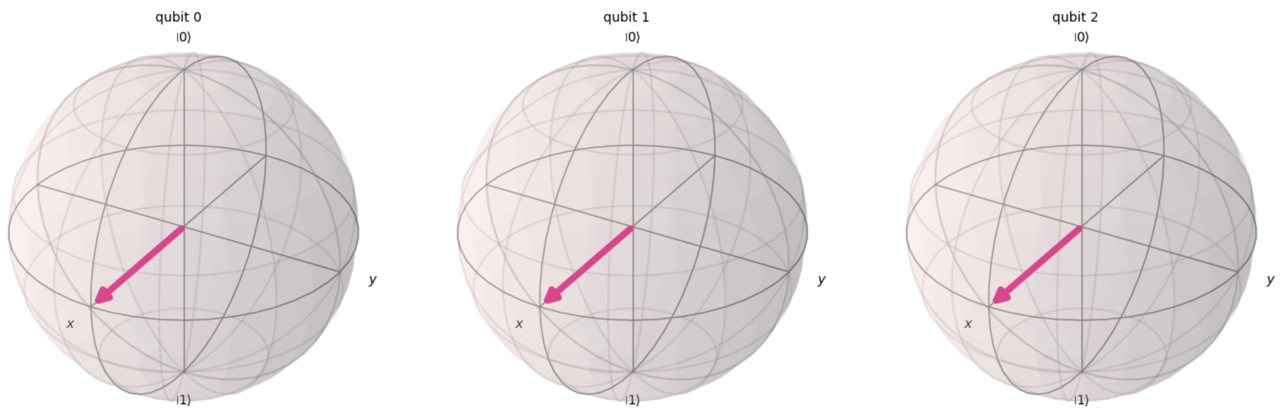
```
state_b = qft_inverse_statevector("011")
plot_bloch_multivector(state_b)
```



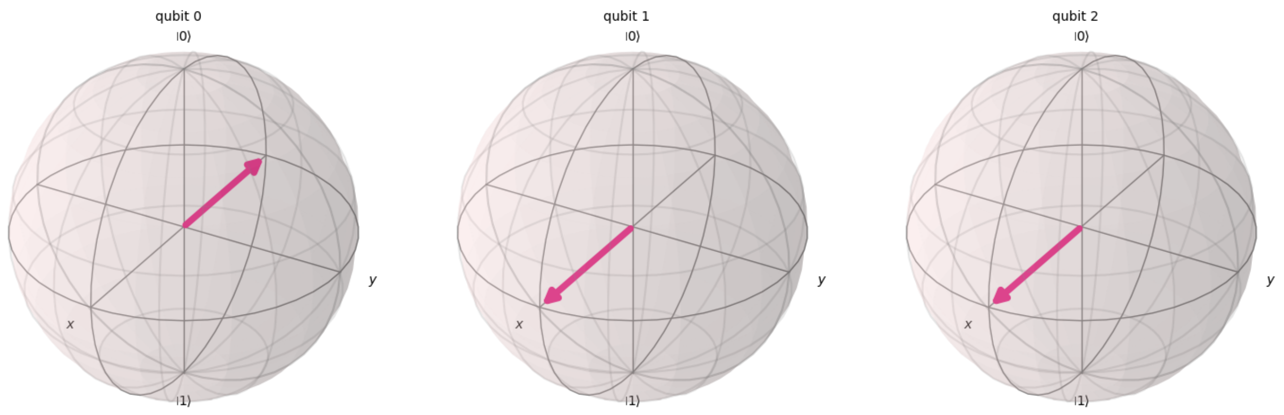
Task 3: Fourier Transform of all 3-qubit basis states

```
for state in ['000','001','010','011','100','101','110','111']:
    qc = encode_big_endian(state)
    qft_big_endian(qc, [0,1,2])
    qc.save_statevector()
    sv = sim.run(qc).result().get_statevector()
    print(f"QFT of |{state}⟩:")
    fig = plot_bloch_multivector(sv)
    fig.savefig(f"outputs/QFT_{state}.png")
```

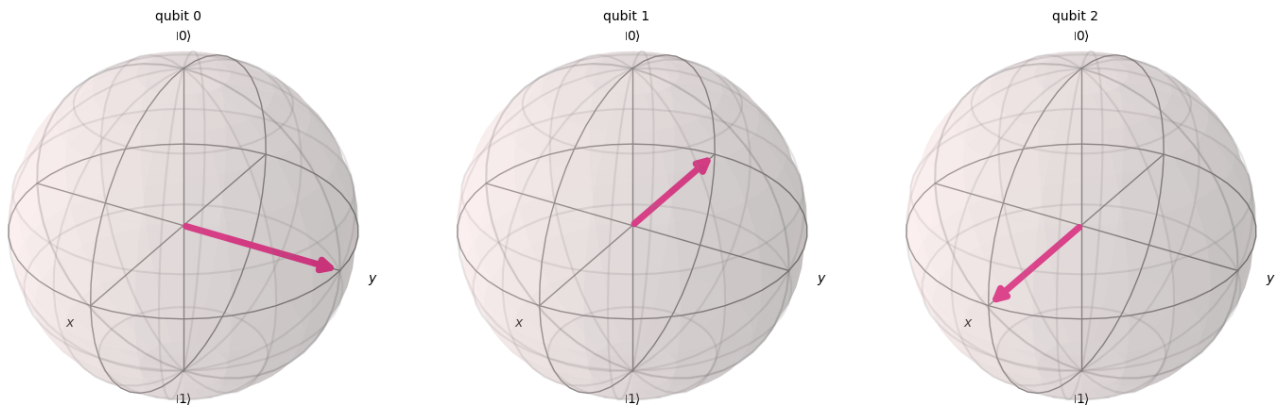
QFT(|000⟩)



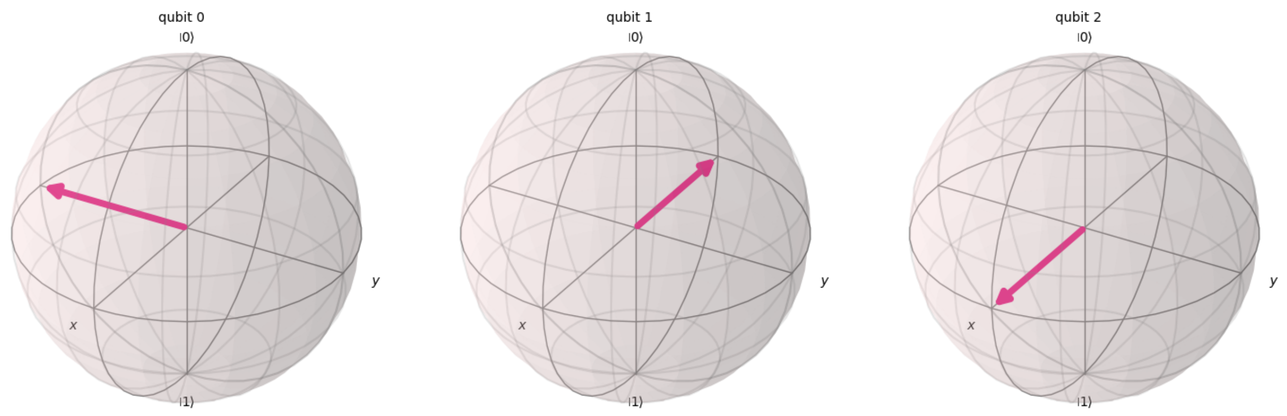
QFT(|001⟩)



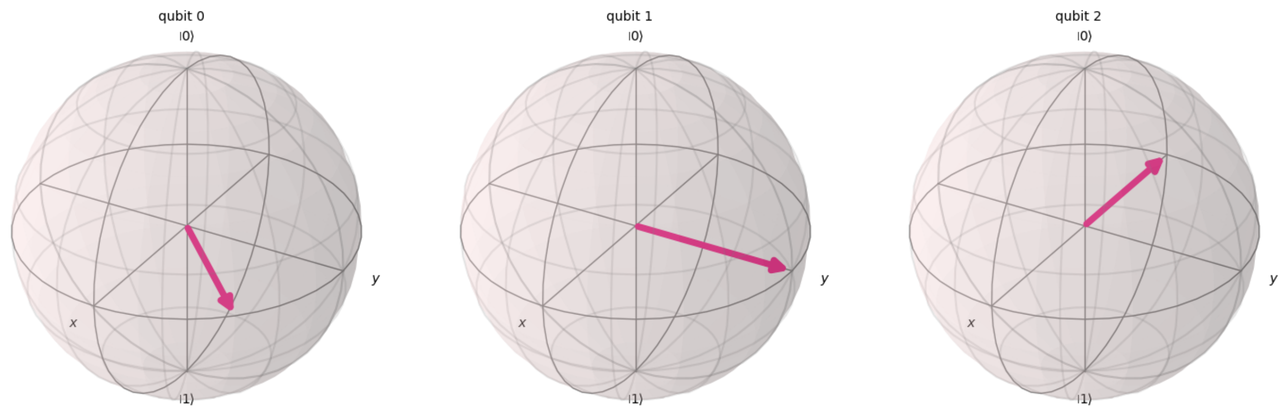
QFT(|010⟩)



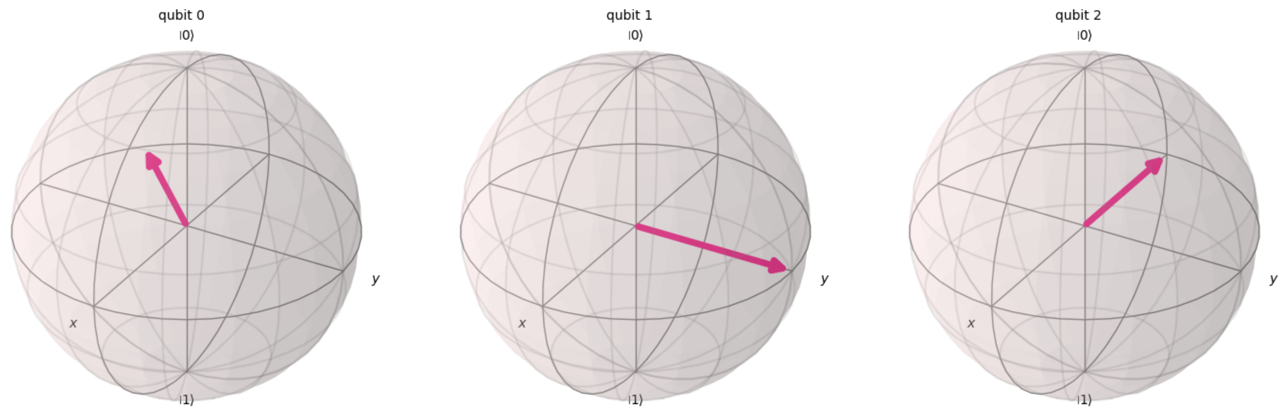
QFT(|011⟩)



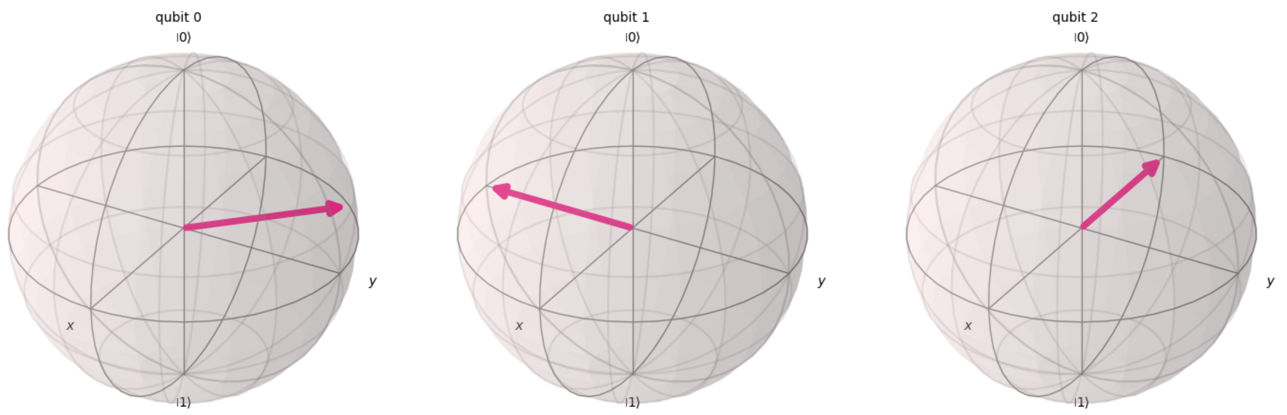
QFT(|100⟩)



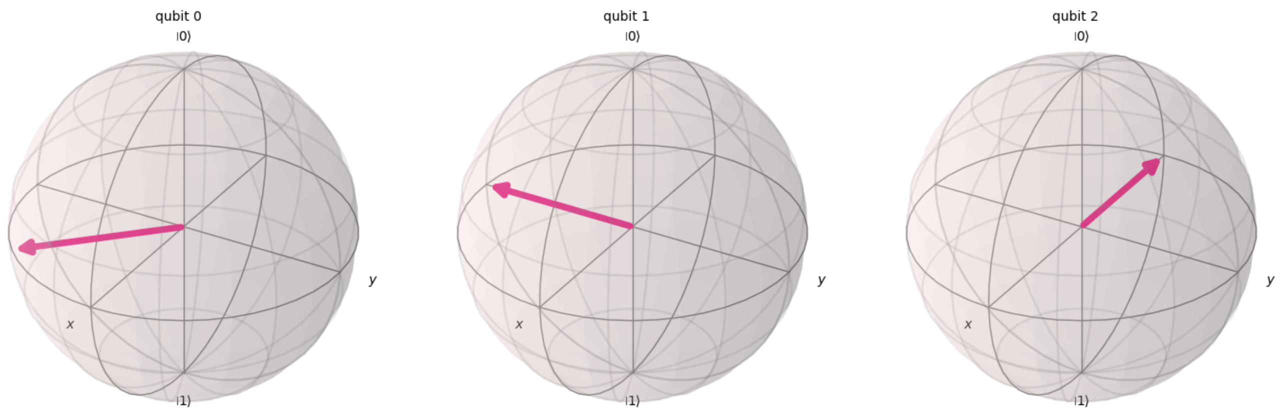
QFT(|101⟩)



QFT(|110⟩)



QFT(|111⟩)



Task 4: Recursive QFT (1–8 qubits)

```
for n_qubits in range(1, 9):
    qc = build_recursive_qft(n_qubits)
    qc.save_statevector()
    sv = sim.run(qc).result().get_statevector()
    print(f"Recursive QFT statevector for {n_qubits} qubits:\n{sv}\n")
```

```
Recursive QFT statevector for 1 qubits:
Statevector([0.70710678+0.j, 0.70710678+0.j],
            dims=(2,))

Recursive QFT statevector for 2 qubits:
Statevector([0.5+0.j, 0.5+0.j, 0.5+0.j, 0.5+0.j],
            dims=(2, 2))

Recursive QFT statevector for 3 qubits:
Statevector([0.35355339+0.j, 0.35355339+0.j, 0.35355339+0.j,
             0.35355339+0.j, 0.35355339+0.j, 0.35355339+0.j,
             0.35355339+0.j, 0.35355339+0.j],
            dims=(2, 2, 2))
```

```
Recursive QFT statevector for 4 qubits:
Statevector([0.25+0.j, 0.25+0.j, 0.25+0.j, 0.25+0.j, 0.25+0.j, 0.25+0.j,
             0.25+0.j, 0.25+0.j, 0.25+0.j, 0.25+0.j, 0.25+0.j, 0.25+0.j,
             0.25+0.j, 0.25+0.j, 0.25+0.j, 0.25+0.j],
            dims=(2, 2, 2, 2))

Recursive QFT statevector for 5 qubits:
Statevector([0.1767767+0.j, 0.1767767+0.j, ..., 0.1767767+0.j],
            dims=(2, 2, 2, 2, 2))

Recursive QFT statevector for 6 qubits:
Statevector([0.125+0.j, 0.125+0.j, ..., 0.125+0.j],
            dims=(2, 2, 2, 2, 2, 2))

Recursive QFT statevector for 7 qubits:
Statevector([0.08838835+0.j, 0.08838835+0.j, ..., 0.08838835+0.j],
            dims=(2, 2, 2, 2, 2, 2, 2))

Recursive QFT statevector for 8 qubits:
Statevector([0.0625+0.j, 0.0625+0.j, ..., 0.0625+0.j],
            dims=(2, 2, 2, 2, 2, 2, 2, 2))
```