Quantum Fourier Transform
Laboratory Report

Riccardo Chimisso 866009

Google Colab Python notebook

Contents

1 Introduction

The aim of the laboratory activity was to explore the circuit implementation of the Quantum Fourier Transform and how it relates to the classical Discrete Fourier Transform.

Four exercises were assigned and are described in the corresponding sections below. The laboratory activity was performed using Google Colab; the complete and polished notebook can be found here.

2 Classical & Quantum Fourier Transform

The **Classical Fourier Transform** (CFT) decomposes a continuous-time signal $f(t)$ into a superposition of complex sinusoids whose frequencies form a continuum.
When the signal is available only at N equally spaced samples, we use the Discrete Fourier Transform (DFT), a linear map

$$X_k = \sum_{n=0}^{N-1} x_n \mathrm{e}^{-\frac{2\pi i k n}{N}}, \qquad k = 0, \ldots, N-1$$

whose naive evaluation costs $\mathcal{O}(N^2)$ complex multiplications (the FFT lowers this to $\mathcal{O}(N \log N)$).

The **Quantum Fourier Transform** (QFT) is the exact analogue executed on a register of $n = \log_2 N$ qubits. Acting on the computational basis state $|x\rangle$ it produces

$$\mathrm{QFT}|x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \mathrm{e}^{\frac{2\pi i x y}{N}} |y\rangle$$

Although its classical input–output relation is identical to the DFT, the latter takes $\mathcal{O}(n2^n)$ gates, whilst the quantum circuit implementation uses only $\mathcal{O}(n^2) =$ gates, an exponential saving that underpins famous quantum algorithms such as Shor's factoring algorithm and the Hidden Subgroup Problem algorithm. It is also used in quantum simulations, quantum error correction, and quantum machine learning.

The QFT can be implemented using a combination of Hadamard gates and controlled phase shift gates. However, the circuit for the QFT can become quite complex as the number of qubits increases, making it challenging to implement in practice.

3 Environment & Prerequisites

All experiments were run in a Google Colab Python notebook (which can be found here).
The required libraries are **PennyLane** for processing quantum circuits and **Matplotlib** for visualization.

It is important to note that, rather than relying on a standalone NumPy installation, the NumPy interface provided by PennyLane is utilized instead. This specialized interface enables automatic differentiation and backpropagation of classical computations through familiar NumPy functions and modules. Additionally, it ensures seamless integration between QNodes and standard NumPy data types, enabling smooth interoperability in hybrid quantum-classical workflows.

4 Exercise 1

4.1 Task

Define the classical Discrete Fourier Transform for vectors with length $N = 2^n$.

4.2 Solution

4.2.1 Code

```
def classical_dft(x):
  N = len(x)
  if N & (N - 1) != 0:
    raise ValueError("Input vector length must be a power of 2.")
  X = np.zeros(N, dtype=np.complex128)
  for k in range(N):
    for n in range(N):
      X[k] += x[n] * np.exp(-2j * pi * k * n / N)
  X /= np.linalg.norm(X)
  return X
```

4.2.2 Comments

It is very easy to see how the code closely corresponds to the calculation of the classical DFT

$$X_k = \sum_{n=0}^{N-1} x_n \mathrm{e}^{-\frac{2\pi i k n}{N}}, \qquad k = 0, \ldots, N-1$$

The only addition being the normalization to scale the results down to the same magnitude as our quantum circuits calculate.
We also force the input length to be a power of 2, again to better align to our quantum circuits.

This code provides a naive implementation of the classical DFT, which can result in substantial computation time for large input vectors. While performance can be greatly enhanced by employing NumPy's optimized FFT, implementing the DFT explicitly has the benefit of being clearer and more intuitive. It reveals the underlying computations step-by-step, offering valuable insight into the fundamental process, rather than hiding it behind a black-box function.

5 Exercise 2

5.1 Task

Define the circuit for the quantum Fourier transform for an arbitrary number n of qubits and print it.

5.2 Solution

5.2.1 Code

```python
def qft_circuit(n_qubits):
  dev = qml.device("default.qubit", wires=n_qubits)

  @qml.qnode(dev)
  def circuit(input_vector):
    if len(input_vector) != 2**n_qubits:
      raise ValueError("Input vector length must be 2^n_qubits.")
    qml.AmplitudeEmbedding(input_vector, range(n_qubits), True)
    for j in range(n_qubits):
      qml.Hadamard(wires=j)
      for k in range(j + 1, n_qubits):
        qml.ControlledPhaseShift(pi * 2**(j - k), wires=[k, j])
    for wire in range(n_qubits // 2):
      qml.SWAP(wires=[wire, n_qubits - 1 - wire])
    return qml.state()

  return circuit

# Print the circuit for 6 qubits.
n = 6
qml.draw_mpl(qft_circuit(n), decimals=2)(np.ones(2**n))
```

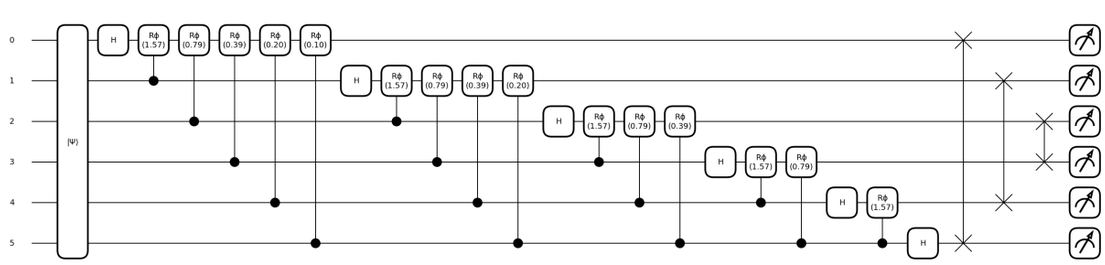

Figure 1: 6-qubit QFT circuit

5.2.2 Comments

The reason why our circuit looks the way it does lies within the mathematical description of the QFT.

The QFT, as shown before, is defined as

$$\mathrm{QFT}|x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \mathrm{e}^{\frac{2\pi i x y}{N}} |y\rangle \tag{1}$$

There are a few considerations to be made to get from that to the equation that defines our circuit.

First, we want to express both x and y in binary form:

$$x = \sum_{j=0}^{n-1} x_j \, 2^j, \qquad y = \sum_{k=0}^{n-1} y_k \, 2^k, \qquad x_j, y_k \in \{0, 1\}$$

Then the phase in (1) becomes

$$\frac{xy}{N} = \frac{1}{2^n} \Big(\sum_j x_j 2^j \Big) \Big(\sum_k y_k 2^k \Big) = \sum_{j,k} x_j y_k \, 2^{j+k-n}$$

Because $j + k - n$ can be negative, we split the sum into integer and fractional parts

$$\frac{xy}{N} = \underbrace{\sum_{j+k \geq n} x_j y_k 2^{j+k-n}}_{\text{integer}} + \underbrace{\sum_{j+k<n} x_j y_k 2^{j+k-n}}_{\text{fraction } 0.***}.$$

The integer part contributes a phase $e^{2\pi i (\text{integer})} = 1$; only the fractional part matters. Plugging the "binary phase" back into the exponent of (1) gives us

$$\mathrm{e}^{\frac{2\pi i x y}{N}} = \mathrm{e}^{2\pi i \sum_{k=0}^{n-1} y_k \, 0.x_k \ldots x_0} = \prod_{k=0}^{n-1} \mathrm{e}^{2\pi i y_k \, 0.x_k \ldots x_0} \tag{2}$$

because each y_k appears only in terms where $j \leq k$.
We can now rewrite the full sum of (1) as an iterated product of single-qubit sums, using (2):

$$\frac{1}{\sqrt{N}} \sum_{y_{n-1}=0}^{1} \cdots \sum_{y_0=0}^{1} \prod_{k=0}^{n-1} \mathrm{e}^{2\pi i y_k \, 0.x_k \ldots x_0} |y_{n-1}\rangle \ldots |y_0\rangle$$

Since the ranges of each y_k are independent, the sums separate:

$$\frac{1}{\sqrt{N}} \bigotimes_{k=0}^{n-1} \Big(\sum_{y_k=0}^{1} \mathrm{e}^{2\pi i y_k \, 0.x_k \ldots x_0} |y_k\rangle \Big)$$

But each inner sum is exactly

$$|0\rangle + \mathrm{e}^{2\pi i \, 0.x_k \ldots x_0} |1\rangle \tag{3}$$

Putting everything together, our final equation becomes

$$\frac{1}{\sqrt{N}} \bigotimes_{k=0}^{n-1} \Big(|0\rangle + \mathrm{e}^{2\pi i \, 0.x_k \ldots x_0} |1\rangle \Big) \tag{4}$$

From (4) it is now clear that each qubit needs a Hadamard gate to enter the superposition $|0\rangle + |1\rangle$, and then we need to apply some phase shift depending on the value of x, which is the input vector.
Looking back at (3), we know that the phase shift we need to apply is a sum of binary fractions of π:

$$\varphi = 2\pi \; 0.x_k \ldots x_0 = 2\pi \sum_{j=k}^{0} x_k 2^{k-j-1} = \pi \sum_{j=k}^{0} x_k 2^{k-j}$$

With these consideration, the circuit (for a registry of 6 qubits) would look like as shown in figure 2.

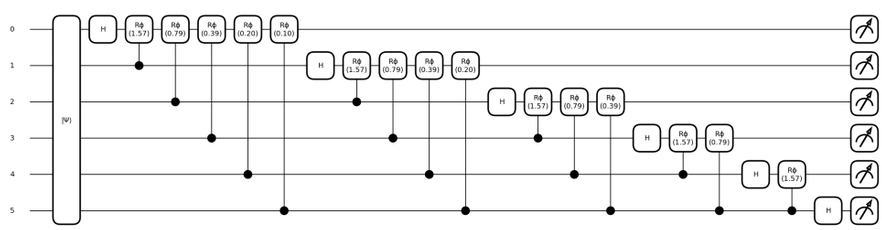

Figure 2: 6-qubit QFT circuit (inverted)

To get to what we see in figures 1 and 3 instead, we need to notice that our gates are applied starting from the least-significant qubit and moving toward the most-significant one. Because of this, qubit 0 ends up holding the highest-order bit of the Fourier-transformed number, qubit 1 holds the next bit, and so on. In other words, after the entangling steps, the qubits are in reverse order with respect to the conventional binary representation.
This is why we apply the final swap gates, so that our circuit complies with the standard binary representation for the DFT.

5.2.3 Additional content

The following is the quantum circuit implementation of the QFT by using PennyLane's native QFT class. It was not required for Exercise 2, but it will turn out that it is necessary for Exercise 3 to compare its results to our own QFT quantum circuit and classical DFT.
Already we can see that the circuit generated with PennyLane's native QFT class looks practically identical to our own.

```
def pl_qft_circuit(n_qubits):
  dev = qml.device("default.qubit", wires=n_qubits)

  @qml.qnode(dev)
  def circuit(input_vector):
    if len(input_vector) != 2**n_qubits:
      raise ValueError("Input vector length must be 2^n_qubits.")
    qml.AmplitudeEmbedding(input_vector, range(n_qubits), True)
    qml.QFT(range(n_qubits))
    return qml.state()

  return circuit

# Print the circuit for 6 qubits.
n = 6
# Decompose the circuit by 1 depth to see the underlying gates for the native QFT.
qml.draw_mpl(qml.transforms.decompose(pl_qft_circuit(n), max_expansion=1),
↪  decimals=2)(np.ones(2**n))
```

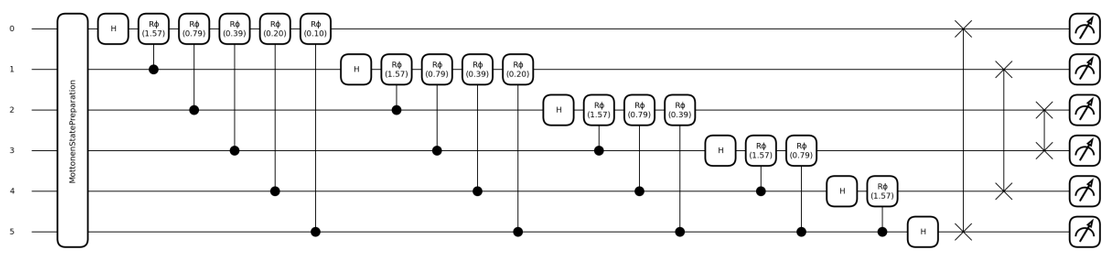

Figure 3: 6-qubit QFT circuit from PennyLane (decomposed by 1 depth)

6 Exercise 3

6.1 Task

Perform the QFT on various input vectors and compare the results with the DFT.
Do this for 4 and 10 qubits. Use an input vector with a single non-zero element, one which is constant and one encoding a period of the function

$$f(t) = 5 + 2cos(2\pi t - \frac{\pi}{2}) + 3cos(4\pi t)$$

Plot the results for the amplitudes of the elements of wave functions after the QFT.
Also compare the results with Pennylane's native QFT class.
Plot the results using Matplotlib.

6.2 Solution

6.2.1 Code

The code for the third exercise has been omitted from this report for the sake of brevity.
To view it, please check out the linked Colab notebook.

6.2.2 Results

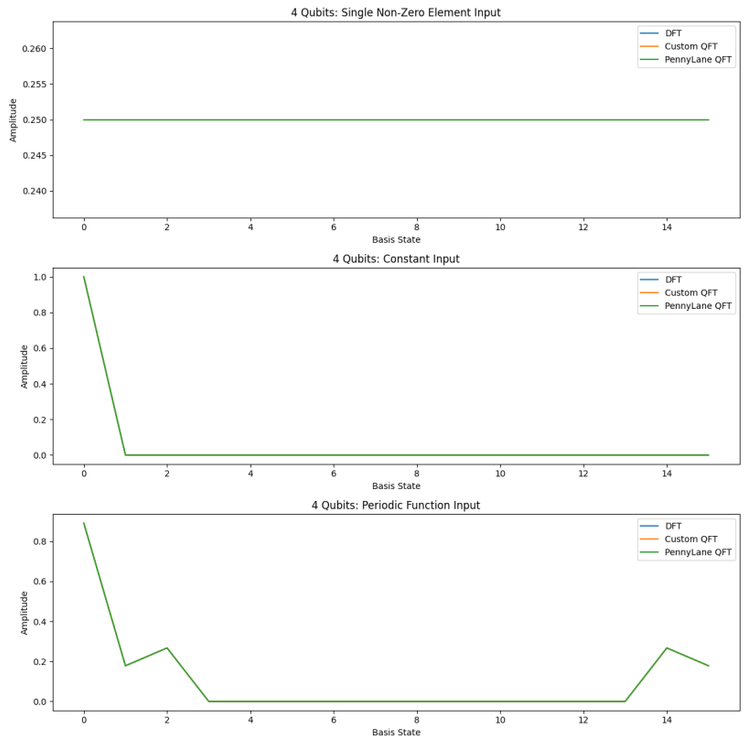

Figure 4: Plots comparing the amplitudes of the classical DFT with both QFT circuits for 4 qubits

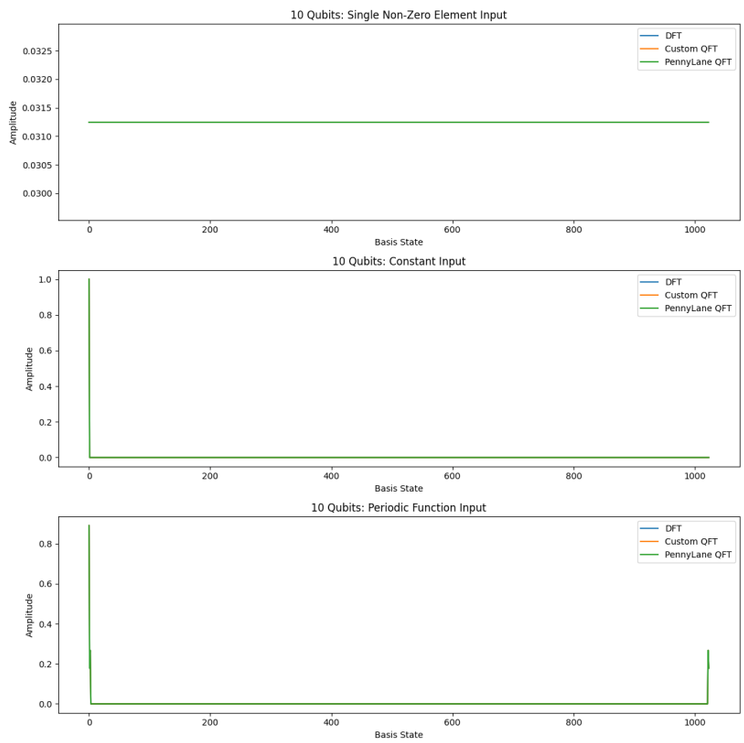

Figure 5: Plots comparing the amplitudes of the classical DFT with both QFT circuits for 10 qubits

The numerical amplitudes were also compared using NumPy's `allclose` function and, as evident from figures 4 and 5, they were all practically the same.

6.2.3 Comments

The results further confirm that our demonstrations and derivations that led us from the classical DFT to the QFT are sound, and that our custom implementations are aligned with the standard implementation from PennyLane.

7 Exercise 4

7.1 Task

Try to guess, by hand, the sequence of elementary gates needed to implement the controlled rotation. Consider that the CNOT exchanges the two last elements of the ket vector.

7.2 Solution

7.2.1 Phase Shift ($P(\varphi)$)

The *phase shift* gate does not change the probability of measuring $|0\rangle$ or $|1\rangle$, however it modifies the phase of the quantum state. It is expressed by the matrix

$$P(\varphi) = \begin{bmatrix} 1 & 0 \\ 0 & \mathrm{e}^{i\varphi} \end{bmatrix}$$

and has the following effect on the 1-qubit computational basis states:

- $|0\rangle \xmapsto{P(\varphi)} |0\rangle$
- $|1\rangle \xmapsto{P(\varphi)} \mathrm{e}^{i\varphi}|1\rangle$

7.2.2 Controlled NOT (CX)

It is expressed by the matrix

$$CX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and has the following effect on the 2-qubit computational basis states:

- $|00\rangle \xmapsto{CX} |00\rangle$
- $|01\rangle \xmapsto{CX} |01\rangle$
- $|10\rangle \xmapsto{CX} |11\rangle$
- $|11\rangle \xmapsto{CX} |10\rangle$

7.2.3 Controlled Phase Shift ($CP(\varphi)$)

With the two gates above, we can find a way to express the *controlled phase shift* gate that acts on the 2-qubit computational basis in the following way:

- $|00\rangle \xmapsto{CP(\varphi)} |00\rangle$
- $|01\rangle \xmapsto{CP(\varphi)} |01\rangle$
- $|10\rangle \xmapsto{CP(\varphi)} |10\rangle$
- $|11\rangle \xmapsto{CP(\varphi)} \mathrm{e}^{i\varphi}|11\rangle$

The above is the result we want to obtain.
It is trivial to see that when the first qubit, the *control* qubit, is $|0\rangle$, the second qubit is left unchanged, while only when the first qubit is $|0\rangle$ the phase shift is applied to the second qubit, the *target* qubit.

7.2.4 Demonstration (Dirac notation)

To get there, we start by first applying a *phase shift* gate of $\frac{\varphi}{2}$ to the first qubit:

- $|00\rangle \xmapsto{P(\frac{\varphi}{2})_0} |00\rangle$
- $|01\rangle \xmapsto{P(\frac{\varphi}{2})_0} |01\rangle$
- $|10\rangle \xmapsto{P(\frac{\varphi}{2})_0} \mathrm{e}^{i\frac{\varphi}{2}}|10\rangle$
- $|11\rangle \xmapsto{P(\frac{\varphi}{2})_0} \mathrm{e}^{i\frac{\varphi}{2}}|11\rangle$

We then apply the *controlled not* gate:

- $|00\rangle \overset{CX}{\longmapsto} |00\rangle$
- $|01\rangle \overset{CX}{\longmapsto} |01\rangle$
- $\mathrm{e}^{i\frac{\varphi}{2}}|10\rangle \overset{CX}{\longmapsto} \mathrm{e}^{i\frac{\varphi}{2}}|11\rangle$
- $\mathrm{e}^{i\frac{\varphi}{2}}|11\rangle \overset{CX}{\longmapsto} \mathrm{e}^{i\frac{\varphi}{2}}|10\rangle$

After, we apply another *phase shift* gate, this time with phase $-\frac{\varphi}{2}$, to the second qubit:

- $|00\rangle \overset{P(-\frac{\varphi}{2})_1}{\longmapsto} |00\rangle$
- $|01\rangle \overset{P(-\frac{\varphi}{2})_1}{\longmapsto} \mathrm{e}^{-i\frac{\varphi}{2}}|01\rangle$
- $\mathrm{e}^{i\frac{\varphi}{2}}|11\rangle \overset{P(-\frac{\varphi}{2})_1}{\longmapsto} |11\rangle$
- $\mathrm{e}^{i\frac{\varphi}{2}}|10\rangle \overset{P(-\frac{\varphi}{2})_1}{\longmapsto} \mathrm{e}^{i\frac{\varphi}{2}}|10\rangle$

Next, we apply again the *controlled not* gate:

- $|00\rangle \overset{CX}{\longmapsto} |00\rangle$
- $\mathrm{e}^{-i\frac{\varphi}{2}}|01\rangle \overset{CX}{\longmapsto} \mathrm{e}^{-i\frac{\varphi}{2}}|01\rangle$
- $|11\rangle \overset{CX}{\longmapsto} |10\rangle$
- $\mathrm{e}^{i\frac{\varphi}{2}}|10\rangle \overset{CX}{\longmapsto} \mathrm{e}^{i\frac{\varphi}{2}}|11\rangle$

Finally, to remove the unwanted phase shift from the second basis state and get the correct phase for the fourth basis state, we just have to apply a *phase shift* gate of $\frac{\varphi}{2}$ to the second qubit:

- $|00\rangle \overset{P(\frac{\varphi}{2})_1}{\longmapsto} |00\rangle$
- $\mathrm{e}^{-i\frac{\varphi}{2}}|01\rangle \overset{P(\frac{\varphi}{2})_1}{\longmapsto} |01\rangle$
- $|10\rangle \overset{P(\frac{\varphi}{2})_1}{\longmapsto} |10\rangle$
- $\mathrm{e}^{i\frac{\varphi}{2}}|11\rangle \overset{P(\frac{\varphi}{2})_1}{\longmapsto} \mathrm{e}^{i\varphi}|11\rangle$

Thus, we demonstrated that by applying that sequence of quantum gates we get an equivalent operation to the *controlled phase shift* with phase φ:

- $|00\rangle \overset{P(\frac{\varphi}{2})_0 \cdot CX \cdot P(-\frac{\varphi}{2})_1 \cdot CX \cdot P(\frac{\varphi}{2})_1}{\longmapsto} |00\rangle$
- $|01\rangle \overset{P(\frac{\varphi}{2})_0 \cdot CX \cdot P(-\frac{\varphi}{2})_1 \cdot CX \cdot P(\frac{\varphi}{2})_1}{\longmapsto} |01\rangle$
- $|10\rangle \overset{P(\frac{\varphi}{2})_0 \cdot CX \cdot P(-\frac{\varphi}{2})_1 \cdot CX \cdot P(\frac{\varphi}{2})_1}{\longmapsto} |10\rangle$
- $|11\rangle \overset{P(\frac{\varphi}{2})_0 \cdot CX \cdot P(-\frac{\varphi}{2})_1 \cdot CX \cdot P(\frac{\varphi}{2})_1}{\longmapsto} \mathrm{e}^{i\varphi}|11\rangle$

7.2.5 Demonstration (Matrix notation)

One can also express the result in the matrix representation. A single-qubit *phase shift* gate $P(\varphi)$ remains well defined in a 2-qubit register, but its 4×4 matrix is less trivial than the 2×2 form shown earlier.
If the phase gate acts on the first qubit, the overall operation is

$$P(\varphi) \otimes I = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\varphi} & 0 \\ 0 & 0 & 0 & e^{i\varphi} \end{pmatrix}$$

Conversely, if it acts on the second qubit, we have

$$I \otimes P(\varphi) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\varphi} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\varphi} \end{pmatrix}$$

We know that the *controlled phase shift* gate has the following matrix form:

$$CP(\varphi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\varphi} \end{pmatrix}$$

Our custom gate sequence ($CGS(\varphi)$) takes the following form instead:

$$CGS(\varphi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\frac{\varphi}{2}} & 0 \\ 0 & 0 & 0 & e^{i\frac{\varphi}{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i\frac{\varphi}{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{-i\frac{\varphi}{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\frac{\varphi}{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\frac{\varphi}{2}} \end{pmatrix}$$

We can solve each matrix multiplication to get the resulting operation:

$$\begin{aligned} CGS(\varphi) &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\frac{\varphi}{2}} & 0 \\ 0 & 0 & 0 & e^{i\frac{\varphi}{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i\frac{\varphi}{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{-i\frac{\varphi}{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\frac{\varphi}{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\frac{\varphi}{2}} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\frac{\varphi}{2}} & 0 \\ 0 & 0 & 0 & e^{i\frac{\varphi}{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i\frac{\varphi}{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{-i\frac{\varphi}{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\frac{\varphi}{2}} & 0 & 0 \\ 0 & 0 & 0 & e^{i\frac{\varphi}{2}} \\ 0 & 0 & 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\frac{\varphi}{2}} & 0 \\ 0 & 0 & 0 & e^{i\frac{\varphi}{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{-i\frac{\varphi}{2}} & 0 \\ 0 & 0 & 0 & e^{i\frac{\varphi}{2}} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\frac{\varphi}{2}} & 0 \\ 0 & 0 & 0 & e^{i\frac{\varphi}{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{-i\frac{\varphi}{2}} & 0 \\ 0 & 0 & 0 & e^{i\frac{\varphi}{2}} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\varphi} \end{pmatrix} \end{aligned}$$

Thus, we successfully demonstrated that

$$CGS(\varphi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\varphi} \end{pmatrix} = CP(\varphi)$$

7.3 Additional content

We can verify the correctness of our demonstrations also by comparing the outcome of applying Pennylane's predefined `ControlledPhaseShift` gate with that obtained from our custom gate sequence.
The following code snippets takes care of such task:

```
# Define the device we will use to compare the results.
dev = qml.device("default.qubit", wires=2)
```

```
# Create a quantum circuit that uses Pennylane's predefined ControlledPhaseShift.
@qml.qnode(dev)
def controlled_phase_shift_circuit(input, phi):
  qml.BasisState(input, wires=[0, 1])
  qml.ControlledPhaseShift(phi, wires=[0, 1])
  return qml.state()

# Print the circuit with an example phase of pi.
qml.draw_mpl(controlled_phase_shift_circuit, decimals=2)([1, 0], pi)
```

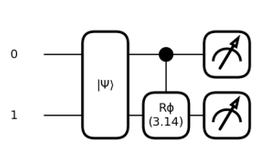

Figure 6: Circuit made with PennyLane's predefined `ControlledPhaseShift`

```python
# Create a quantum circuit that uses our custom gate sequence.
@qml.qnode(dev)
def controlled_phase_shift_circuit_decomposed(input, phi):
  qml.BasisState(input, wires=[0, 1])
  qml.PhaseShift(phi/2, wires=0)
  qml.CNOT(wires=[0, 1])
  qml.PhaseShift(-phi/2, wires=1)
  qml.CNOT(wires=[0, 1])
  qml.PhaseShift(phi/2, wires=1)
  return qml.state()

# Print the circuit with an example phase of pi.
qml.draw_mpl(controlled_phase_shift_circuit_decomposed, decimals=2)([1, 0], pi)
```

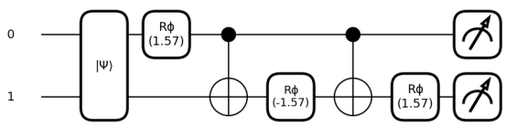

Figure 7: Circuit made using our custom gate sequence

```python
# Compare the results for each basis state with different test phase shifts.
for phi in [0, pi/2, pi, 3*pi/2, 2*pi]:
  print(f"Phase Shift: {phi:.4f}")
  for q0 in range(2):
    for q1 in range(2):
      print(f"Basis State |{q0}{q1}⟩: {np.allclose(controlled_phase_shift_circuit([q0, q1], pi),
      ↪ controlled_phase_shift_circuit_decomposed([q0, q1], pi))}")
```

The code snippet right above produces the output summarized in table 1.

φ (rad)	$\|00\rangle$	$\|01\rangle$	$\|10\rangle$	$\|11\rangle$
0.0000	✓	✓	✓	✓
1.5708	✓	✓	✓	✓
3.1416	✓	✓	✓	✓
4.7124	✓	✓	✓	✓
6.2832	✓	✓	✓	✓

Table 1: Equivalence check between PennyLane's `ControlledPhaseShift` and our custom gate sequence across multiple phase angles

8 Conclusions

The four laboratory exercises collectively confirm that the quantum–computational implementation of the Fourier transform behaves exactly as predicted by theory.

- **Exercise 1** reproduced the classical DFT from first principles, laying a numerical baseline for all subsequent comparisons.
- **Exercise 2** developed a gate-level QFT circuit whose structure follows directly from the binary-fraction decomposition of the phase, and matched PennyLane's built-in `QFT`.
- **Exercise 3** showed, for 4- and 10-qubit systems, that the amplitude spectra produced by the classical DFT, the custom QFT, and PennyLane's native QFT are indistinguishable within numerical precision, even for non-trivial input signals.
- **Exercise 4** derived and experimentally verified an elementary gate sequence that performs a controlled-phase operation; matrix algebra and state-vector tests confirmed its equivalence to `ControlledPhaseShift` across a full 2π sweep of phases.

However, it is important to note that QFT circuit depth grows quadratically with qubit count: even the 6-qubit example already needs a ladder of 15 controlled phase shift gates, and every extra qubit adds another rung. The gate count increases even more as we decompose controlled phase shift gates into elementary ones and take into account the additional Hadamard and swap gates. All of this makes practical implementations of the QFT challenging.