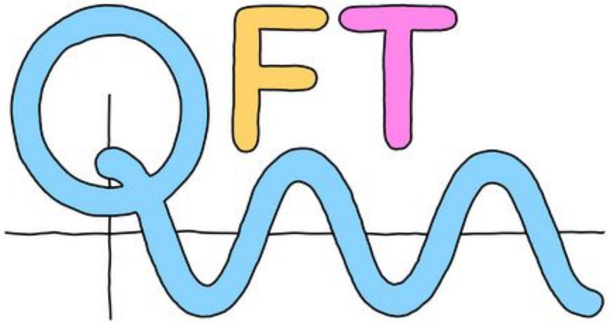
Introduction to Quantum Computing Pennylane: Quantum Fourier transform

Chris Heunen



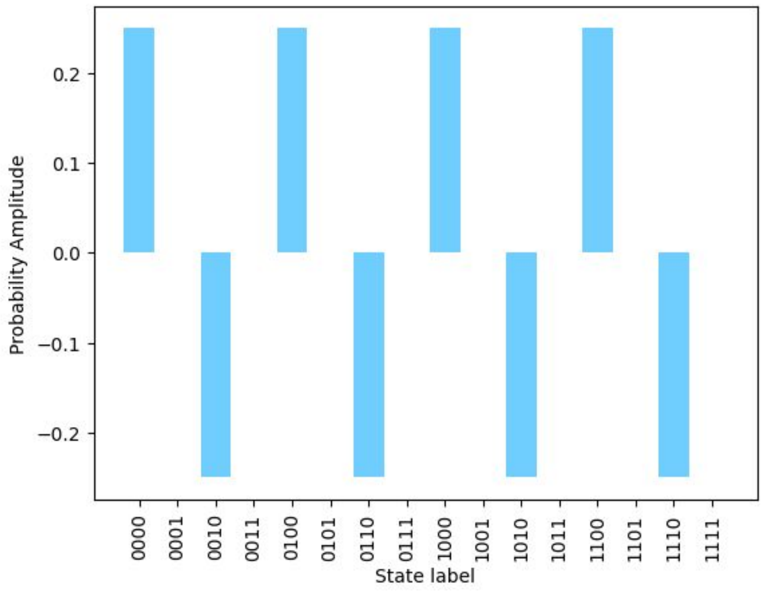
Exercise 1: The QFT matrix

$$\frac{1}{\sqrt{N}}\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix}$$

```
def matrix():
    N = 2 ** n_qubits
    m = np.empty([N,N]).astype(complex)
    # define the QFT matrix
    return m

print(matrix())
```

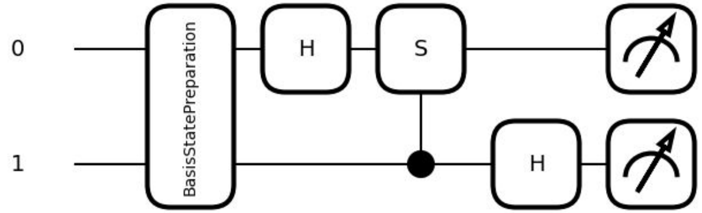
Exercise 1: The QFT matrix

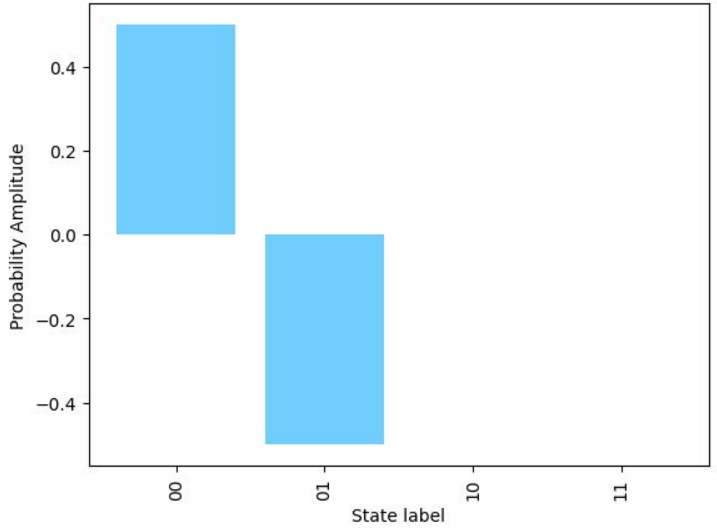
```
@pl.qnode(dev)
def circuit(k):
    bits = [int(x) for x in np.binary_repr(k, width=n_qubits)]
    print(bits)
    pl.BasisStatePreparation(bits, wires=wires)
    pl.QubitUnitary(matrix(), wires=wires)
    return pl.state()

results = pl.snapshots(circuit)(3)
y = np.real(results["execution_results"])
bit_strings = [f"{x:0{n_qubits}b}" for x in range(len(y))]
plt.bar(bit_strings, y, color = "#70CEFF")
plt.xticks(rotation="vertical")
plt.xlabel("State label")
plt.ylabel("Probability Amplitude")
plt.show()
```



Exercise 2: The 2-qubit QFT circuit

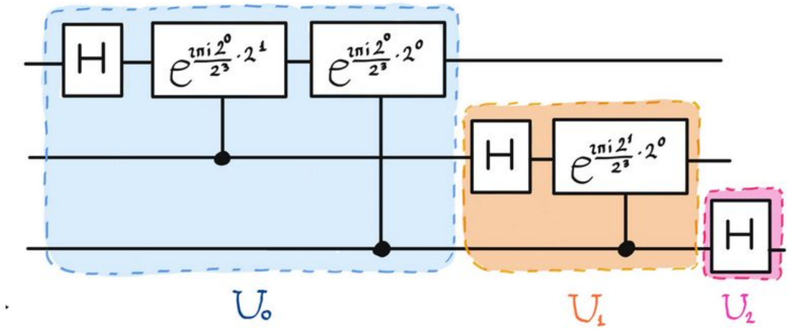
```
n_qubits = 2
dev = pl.device("default.qubit", wires=n_qubits)
wires = list(range(n_qubits))

@pl.qnode(dev)
def circuit(k):
    bits = [int(x) for x in np.binary_repr(k, width=n_qubits)]
    pl.BasisStatePreparation(bits, wires=wires)
    # Build the 2-qubit QFT circuit
    return pl.state()

pl.draw_mpl(circuit, style="black_white")(1);

# Plot the results, compare against matrix
```



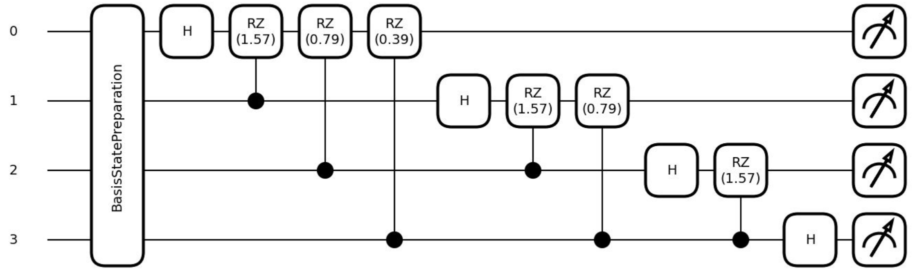


Exercise 3: The recursive QFT circuit

```
n_qubits = 4
dev = pl.device("default.qubit", wires=n_qubits)
wires = list(range(n_qubits))

def subcircuit(wires):
   # build the subcircuit on the given wires

def QFT():
   # put all the subcircuits together

@pl.qnode(dev)
def circuit(k):
   bits = [int(x) for x in np.binary_repr(k,
width=n_qubits)]
   pl.BasisStatePreparation(bits, wires=wires)
   QFT()
   return pl.state()

pl.draw_mpl(circuit, style="black_white", decimals=2)(3);
# plot the results
```

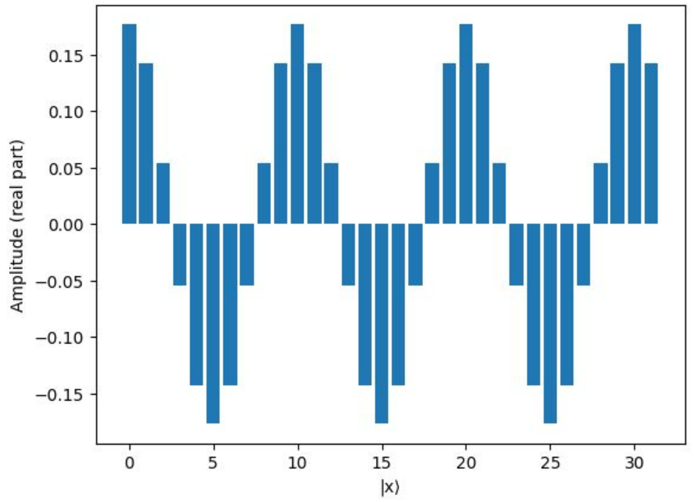


H
$e^{\frac{2\pi i 2^0}{2^3}\cdot 2^1}$
$e^{\frac{2\pi i 2^0}{2^3}\cdot 2^0}$
H
$e^{\frac{2\pi i 2^1}{2^3}\cdot 2^0}$
H
U_0
U_1
U_2



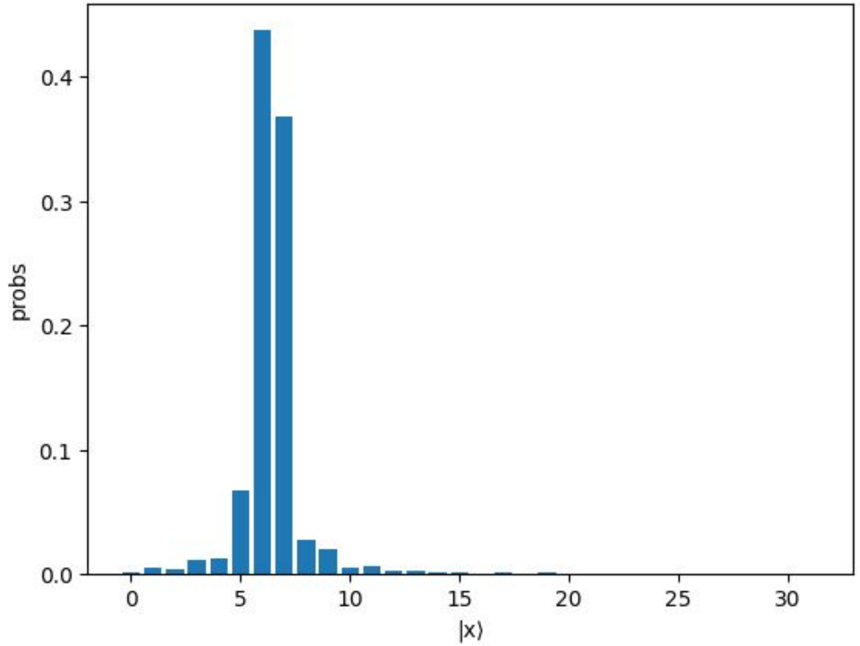
Exercise 4: Using the QFT for period-finding

```
n_qubits = 6
period = 10
dev = pl.device("default.qubit", wires=n_qubits)
wires = list(range(n_qubits))

def periodicstatecircuit():
   # set up a circuit that output a periodic state,
using pl.ControlledSequence and pl.PhaseShift

@pl.qnode(dev)
def periodicstate():
   periodicstatecircuit()
   return pl.state()

state = periodicstate().real[:2**(n_qubits-1)]
# plot the results
```



Exercise 4: Using the QFT for period-finding

```
@pl.qnode(dev)
def circuit():
 periodicstatecircuit()
 pl.QFT(wires=wires)
 return pl.probs(wires=wires)

state = circuit()[:2**(n_qubits-1)]
# plot the results

# Find the most probable answer and translate that
in the the most likely period
print(2**(n_qubits)/6)
```


0.4
0.3
0.2
0.1
0.0
probs
0
5
10
15
20
25
30
|x⟩