2025 / Spring Semester

Topics on Quantum Computing

Lecture Note – Quantum Fourier Transform

Junghee Ryu

KISTI

Quantum Fourier Transform (QFT)

A counterpart of classical Discrete Fourier Transform

- One of the most important building blocks in quantum algorithms, famously used in quantum phase estimation, Shor's factoring algorithm, and more.
- A quantum analog of the discrete Fourier transform (DFT)
  - DFT is the main tool of digital signal processing
  - Typically employed to analyze periodic functions by mapping between time and frequency representations.
- DFT
  - input vector: $(x_0, \ldots, x_{N-1}) \in \mathbb{C}^N$
  - output vector: $(y_0, \ldots, y_{N-1}) \in \mathbb{C}^N$

$$y_k = \sum_{j=0}^{N-1} x_j \exp\left(-\frac{2\pi i k j}{N}\right)$$

Quantum Fourier Transform (QFT)

A counterpart of classical Discrete Fourier Transform

- The idea of the QFT is to perform the same operation but to a quantum state $|x\rangle = \sum_{i=0}^{N-1} x_i |i\rangle$ to get a quantum state $|y\rangle = \sum_{i=0}^{N-1} y_i |i\rangle$ as follows

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \exp\left(\frac{2\pi ikj}{N}\right) \Rightarrow |j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle$$

  - Input and output state are normalized ones: Scaler (1/sqrt(N)) is necessary
  - $N = 2^n$, where n indicates the size of a quantum circuit involving QFT
  - Strictly speaking, DFT corresponds to QFT^{+}

Quantum Fourier Transform (QFT)

Circuit representation

Basis $\qquad$ Qubit $\qquad$ (for $n=2$)

$|0\rangle \longrightarrow |00\rangle$

$|1\rangle \longrightarrow |01\rangle$

$|2\rangle \longrightarrow |10\rangle$

$|3\rangle \longrightarrow |11\rangle$

- ❑ Basis Representation
- ❑ Qubit Representation

Quantum Fourier Transform (QFT)

Circuit representation

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k/N} |k\rangle$$

Binary Representation

$$j = j_1 j_2 \dots j_n$$
$$= j_1 2^{n-1} + j_2 2^{n-2} + \dots + j_n 2^0$$

Binary Fraction

$$0.j_l j_{l+1} \dots j_m$$
$$= j_l/2 + j_{l+1}/4 + \dots + j_m/2^{m-l+1}$$

$$|j_1, \dots, j_n\rangle \longrightarrow \frac{1}{2^{n/2}} \times$$

$$\left(|0\rangle + e^{2\pi i 0.j_n}|1\rangle\right) \otimes$$

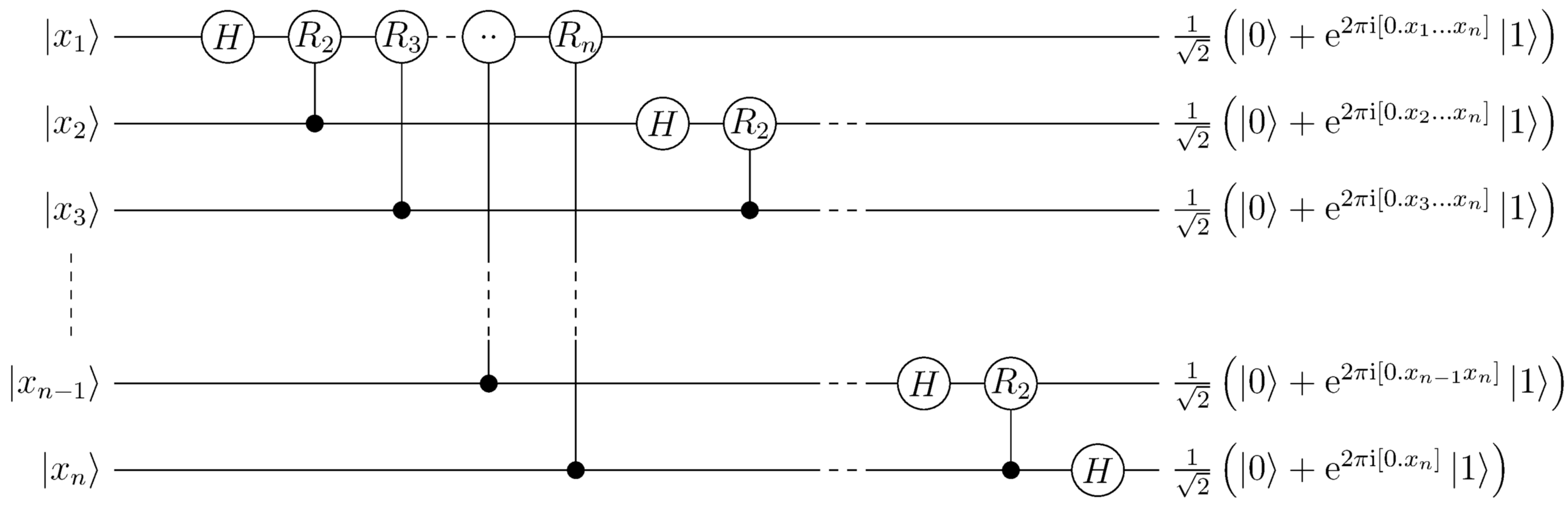
$$\left(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle\right) \otimes$$

$$\cdots \otimes$$

$$\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \cdots j_n}|1\rangle\right)$$

Quantum Fourier Transform (QFT)

Circuit representation

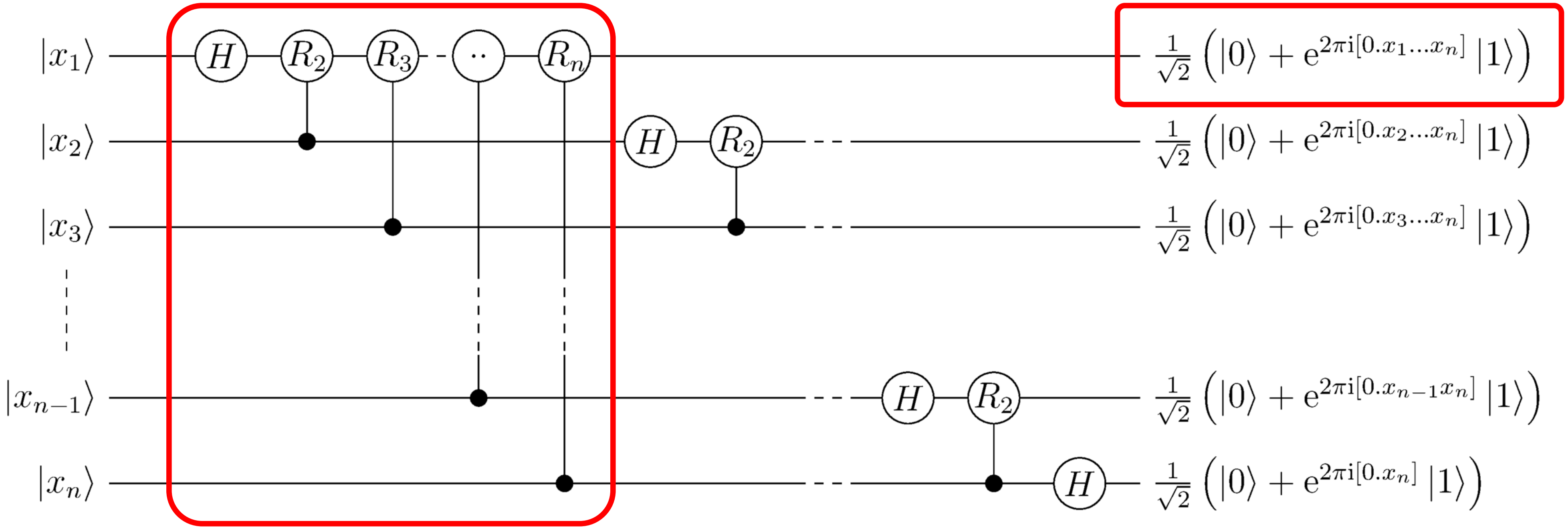


$$|x\rangle = |x_1 x_2 \ldots x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \cdots \otimes |x_n\rangle$$

$$\rightarrow \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i[0.x_1 \ldots x_n]}|1\rangle\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i[0.x_2 \ldots x_n]}|1\rangle\right) \otimes \ \ldots \ \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i[0.x_n]}|1\rangle\right)$$

Quantum Fourier Transform (QFT)

Circuit representation

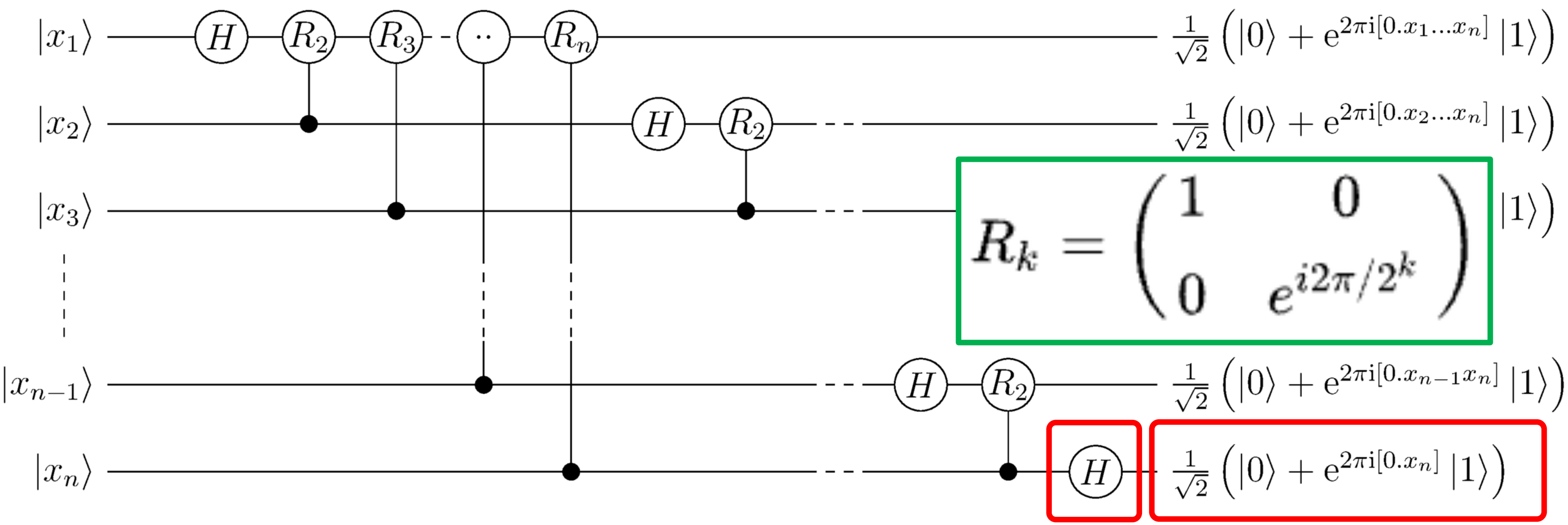


$$|x\rangle = |x_1 x_2 \dots x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \cdots \otimes |x_n\rangle$$

$$\rightarrow \boxed{\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i[0.x_1 \dots x_n]}|1\rangle\right)} \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i[0.x_2 \dots x_n]}|1\rangle\right) \otimes \dots \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i[0.x_n]}|1\rangle\right)$$

Quantum Fourier Transform (QFT)

Circuit representation



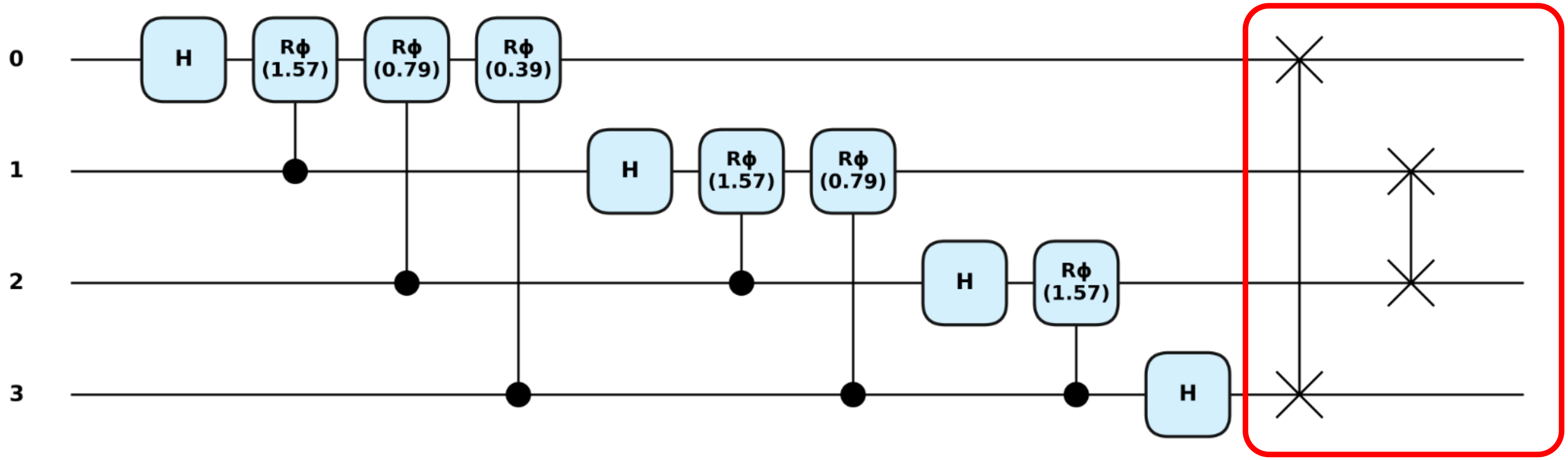
$$|x\rangle = |x_1 x_2 \ldots x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \cdots \otimes |x_n\rangle$$

$$\rightarrow \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i[0.x_1\ldots x_n]}|1\rangle\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i[0.x_2\ldots x_n]}|1\rangle\right) \otimes \ \ldots \ \otimes \boxed{\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i[0.x_n]}|1\rangle\right)}$$

Quantum Fourier Transform (QFT)

Circuit representation

❑ 4-qubit QFT circuit



Why do we need the last block of SWAP gates?

Quantum Fourier Transform (QFT)

Starting with a simple programming example

```
from scipy.linalg import dft
import pennylane as qml
import numpy as np

n = 2

print("DFT matrix for n = 2:\n")
print(np.round(1 / np.sqrt(2 ** n) * dft(2 ** n), 2))

qft_inverse = qml.adjoint(qml.QFT([0,1]))

print("\n inverse QFT matrix for n = 2:\n")
print(np.round(qft_inverse.matrix(), 2))
```

```
DFT matrix for n = 2:

[[ 0.5+0.j  0.5+0.j  0.5+0.j  0.5+0.j ]
 [ 0.5+0.j  0.  -0.5j -0.5-0.j -0.  +0.5j]
 [ 0.5+0.j -0.5-0.j  0.5+0.j -0.5-0.j ]
 [ 0.5+0.j -0.  +0.5j -0.5-0.j  0.  -0.5j]]

inverse QFT matrix for n = 2:

[[ 0.5-0.j  0.5-0.j  0.5-0.j  0.5-0.j ]
 [ 0.5-0.j  0.  -0.5j -0.5-0.j  0.  +0.5j]
 [ 0.5-0.j -0.5-0.j  0.5-0.j -0.5-0.j ]
 [ 0.5-0.j  0.  +0.5j -0.5-0.j  0.  -0.5j]]
```

- ❑ `scipy.linalg` → `dft` : return a DFT matrix
- ❑ `adjoint`: equivalent to the inverse operation for unitary matrices

Quantum Fourier Transform (QFT)

Starting with a simple programming example

```
import pennylane as qml
import numpy as np
import matplotlib.pyplot as plt

dev = qml.device("default.qubit", wires=2)

@qml.qnode(dev)
def generate_bellstate():
        qml.Hadamard(wires=1)
        qml.CNOT(wires=[0,1])
        return qml.state()

generate_bellstate()

@qml.qnode(dev)
def doQFT():
        qml.QFT(wires=[0,1])
        return qml.state()

doQFT()
```

- ❑ `@qml.qnode(device name)`
  - Specify the device where a circuit function will be executed
  - Should be always known before we define every circuit functions
- ❑ State is not given as an explicit variable
  - It’s implicitly evolving with function calls
  - Vector elements of the state can be checked with `qml.state()`→ only in emulator (why?)
  - `qml.probs()`is more general in quantum information process

Quantum Fourier Transform (QFT)

Starting with a simple programming example

```
import pennylane as qml
import numpy as np
import matplotlib.pyplot as plt

dev = qml.device("default.qubit", wires=2)

@qml.qnode(dev)
def generate_bellstate():
        qml.Hadamard(wires=1)
        qml.CNOT(wires=[0,1])
        return qml.probs([0,1])

generate_bellstate()

@qml.qnode(dev)
def doQFT():
        qml.QFT(wires=[0,1])
        return qml.probs([0,1])

doQFT()
```

- `@qml.qnode(device name)`
  - Specify the device where a circuit function will be executed
  - Should be always known before we define every circuit functions
- State is not given as an explicit variable
  - It's implicitly evolving with function calls
  - Vector elements of the state can be checked with `qml.state()` → only in emulator (why?)
  - `qml.probs()` is more general in quantum information process

Quantum Fourier Transform (QFT)

Starting with a simple programming example

```
import pennylane as qml
import numpy as np
import matplotlib.pyplot as plt

dev = qml.device("default.qubit", wires=2, shots=1024)

@qml.qnode(dev)
def generate_bellstate():
        qml.Hadamard(wires=1)
        qml.CNOT(wires=[0,1])
        return qml.counts()

generate_bellstate()

@qml.qnode(dev)
def doQFT():
        qml.QFT(wires=[0, 1])
        return qml.counts()

doQFT()
```

- Shots & measurements
  - Recall what you really get from the quantum computation
  - shots & counts
- What happens If you use `qml.counts(qml.Z(0))`?
  - What do the results mean?

Quantum Fourier Transform (QFT)

More realistic practice

- Purpose of QFT: Why do we use the Fourier Transform?
  - Many answers would be possible: One of primary reasons is to approximate or find the period of a given function (Time & Spatial ←→ Frequency)
- Practice
  - Find the period of a given 5-qubit signal using QFT
  - Let's imagine we have an operator that prepares the signal (input state) whose associated period is 10

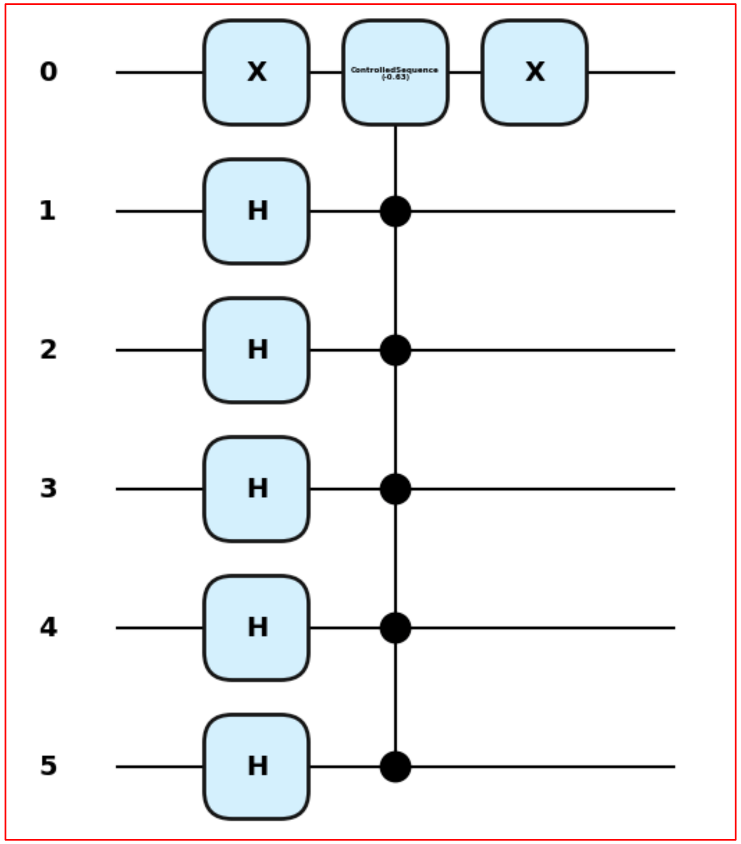
$$|\psi\rangle = \frac{1}{\sqrt{2^5}} \sum_{x=0}^{31} \exp\left(\frac{-2\pi i x}{10}\right) |x\rangle,$$

Quantum Fourier Transform (QFT)

Generate the desired input state with PENNYLANE

```
import pennylane as qml
import numpy as np
import matplotlib.pyplot as plt

def prep():
    qml.PauliX(wires=0)
    for wire in range(1,6):
        qml.Hadamard(wires=wire)
    qml.ControlledSequence(qml.PhaseShift(-2*np.pi/10, wires=0)
    ..., control=range(1,6))
    qml.PauliX(wires=0)

qml.draw_mpl(prep, decimals = 2, style = "pennylane")()
plt.show()
```

$$|\psi\rangle = \frac{1}{\sqrt{2^5}} \sum_{x=0}^{31} \exp\left(\frac{-2\pi i x}{10}\right)|x\rangle,$$

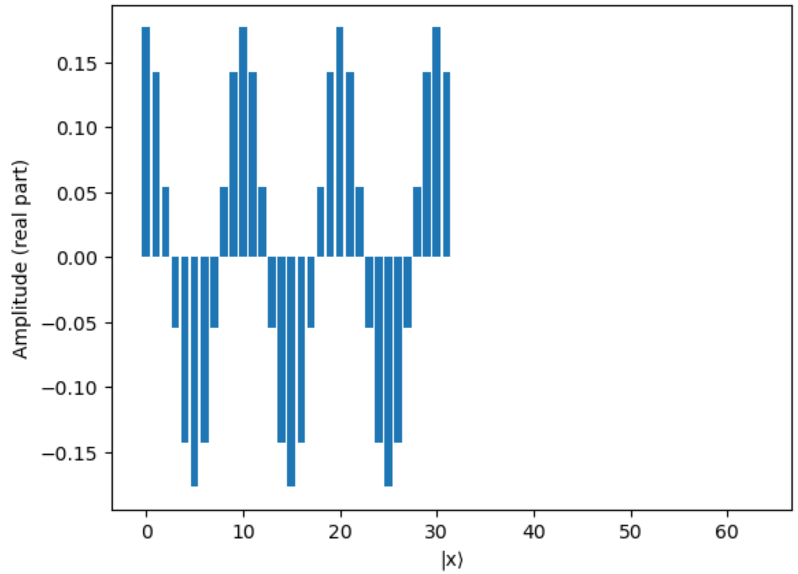


Quantum Fourier Transform (QFT)

Generate the desired input state with PENNYLANE

```
import pennylane as qml
import numpy as np
import matplotlib.pyplot as plt

def prep():
    qml.PauliX(wires=0)
    for wire in range(1,6):
        qml.Hadamard(wires=wire)
    qml.ControlledSequence(qml.PhaseShift(-2*np.pi/10, wires=0)
    ..., control=range(1,6))
    qml.PauliX(wires=0)

dev = qml.device("default.qubit")
@qml.qnode(dev)
def circuit1():
    prep()
    return qml.state()
```

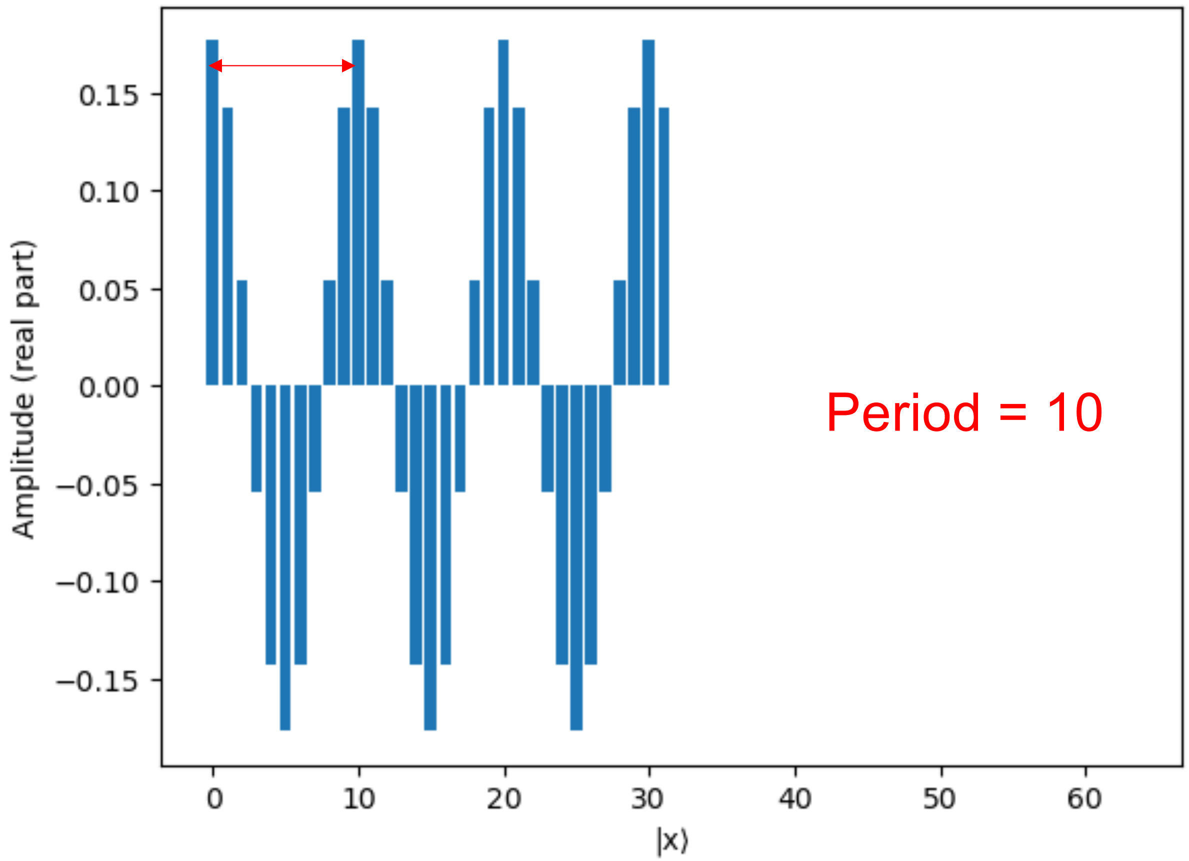
```
state = circuit().real[:32]

plt.bar(range(len(state)), state)
plt.xlabel("|x⟩")
plt.ylabel("Amplitude (real part)")
plt.show()
```



Quantum Fourier Transform (QFT)

Generate the desired input state with PENNYLANE

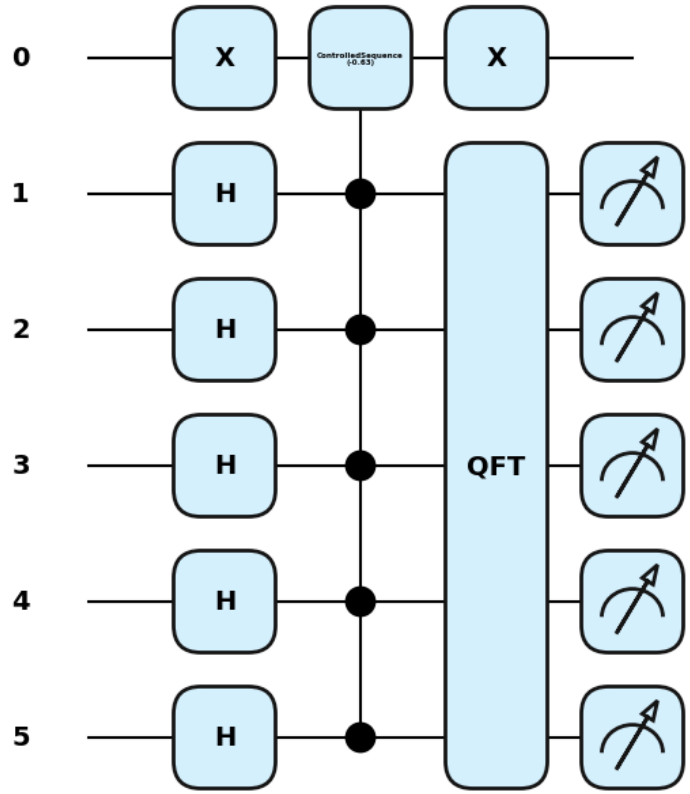
```
import pennylane as qml
import numpy as np
import matplotlib.pyplot as plt

def prep():
    qml.PauliX(wires=0)
    for wire in range(1,6):
        qml.Hadamard(wires=wire)
    qml.ControlledSequence(qml.PhaseShift(-2*
    ..., control=range(1,6))
    qml.PauliX(wires=0)

dev = qml.device("default.qubit")
@qml.qnode(dev)
def circuit1():
    prep()
    return qml.state()
```



Quantum Fourier Transform (QFT)

Take QFT to the generated input state

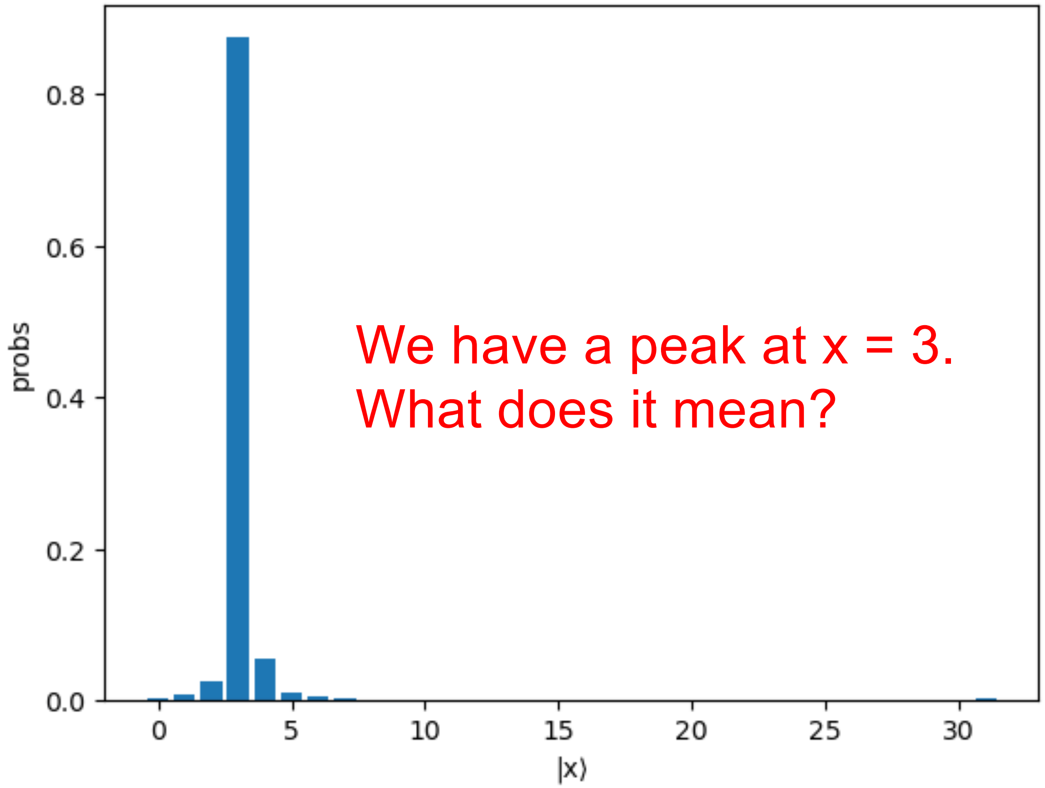
```
...

@qml.qnode(dev)
def circuit2():
    prep()
    qml.QFT(wires=range(1,6))
    return qml.probs(wires=range(1,6))

qml.draw_mpl(circuit2, decimals = 2, style = "pennylane")()
plt.show()

state = circuit2()

plt.bar(range(len(state)), state)
plt.xlabel("|x⟩")
plt.ylabel("probs")
plt.show()
```



Quantum Fourier Transform (QFT)

Take QFT to the generated input state

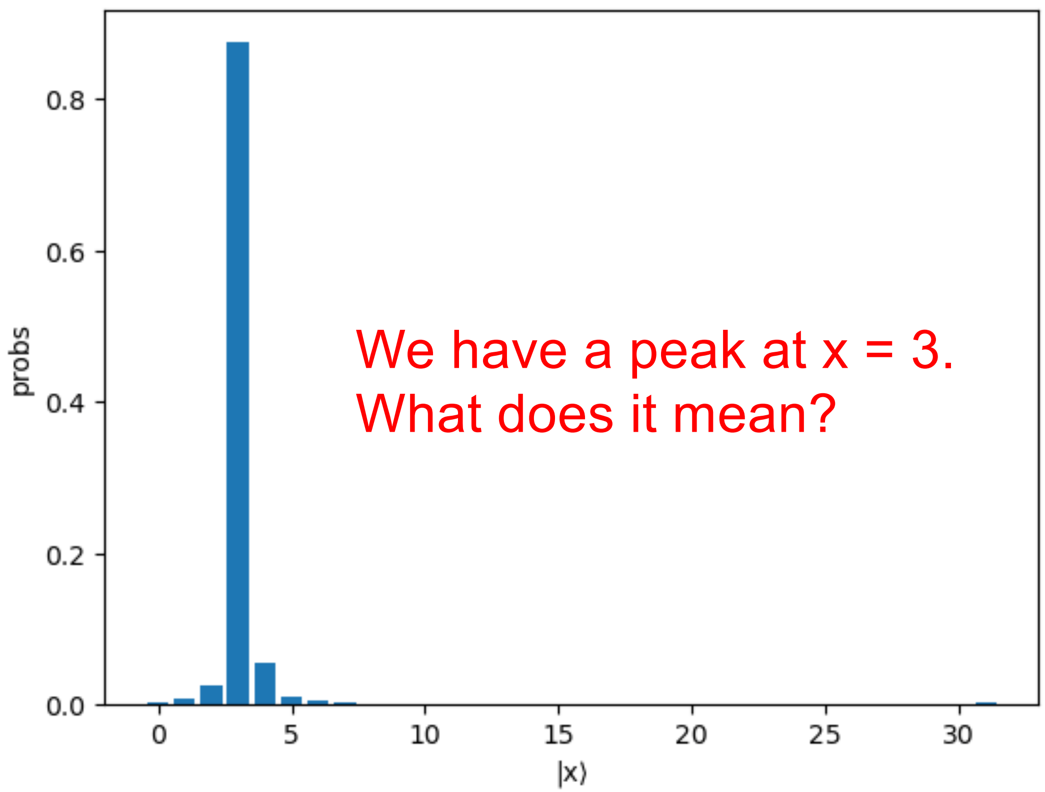
```
...

@qml.qnode(dev)
def circuit2():
    prep()
    qml.QFT(wires=range(1,6))
    return qml.probs(wires=range(1,6))

qml.draw_mpl(circuit2, decimals = 2, style = "penny

plt.show()

state = circuit2()

plt.bar(range(len(state)), state)
plt.xlabel("|x⟩")
plt.ylabel("probs")
plt.show()
```



Quantum Fourier Transform (QFT)

Take QFT to the generated input state

- The peak value @ x = 3
  - x ~ 2^n x f
  - n = qubit sizes (2^n = # of samples)
  - f = frequency (1/Period)
- Interpretation of the result
  - x = 32 x f
  - f = 3/32 → Period = 1/f = 32/3
    ~ 10.66, being close to 10
  - Question
    → Why do we get 10.66 not 10?



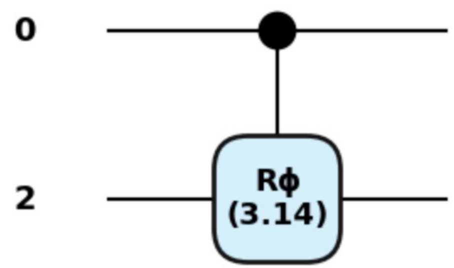
Quantum Fourier Transform (QFT)

Misson

- ❑ Write a subroutine named as “QFT_inhouse” that satisfies following conditions
  - Handles a 10-qubit problem
  - All the gate operations must be programmed by manually
  - Recall the circuit representation shown in slide 8 & 9
  - You will need the `ControlledPhaseShift` routine in PENNYLANE

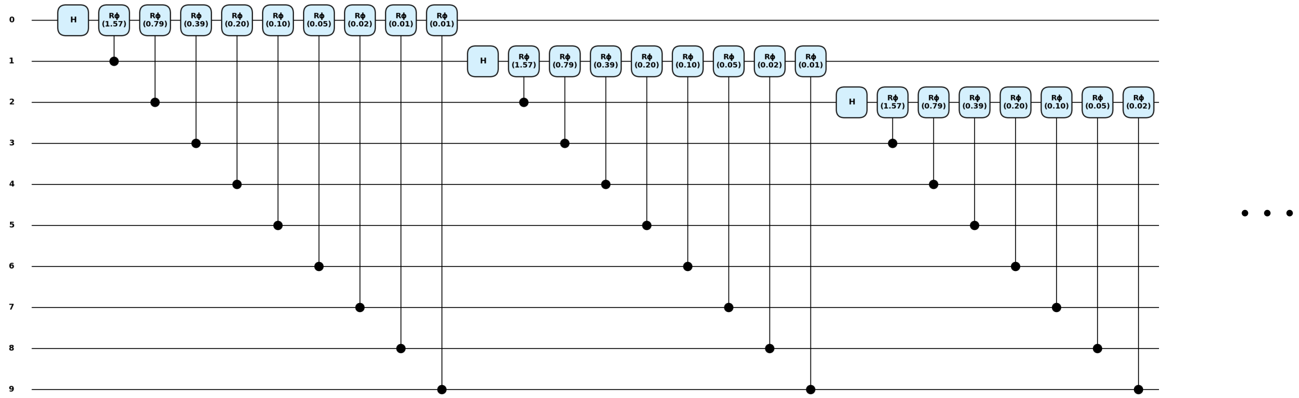
    → `qml.ControlledPhaseShirt(PHI, wires=[control,target])`

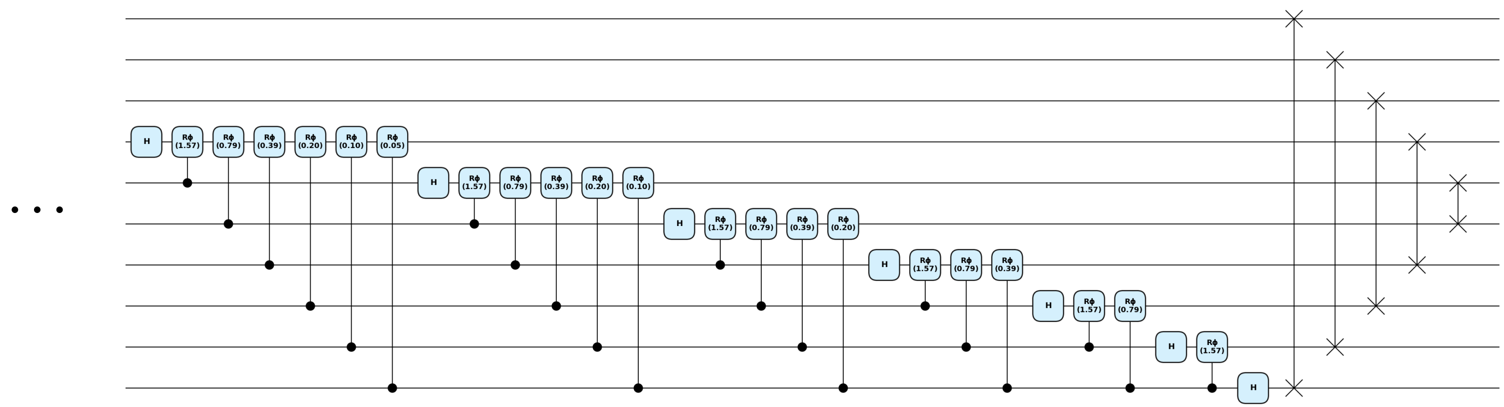
```
def circuit3():
        qml.ControlledPhaseShift(np.pi, wires=[0,2])
qml.draw_mpl(circuit3, decimals = 2, style = "pennylane")()
plt.show()
```



Quantum Fourier Transform (QFT)

Misson





- ❑ Can you generalize your subroutine for arbitrary circuit sizes (n qubits)?