第七章
Shor 演算法量子程式設計

國立中央大學 資工系
江振瑞

大綱

- 基本概念
- 量子傅立葉變換
- 量子相位估測
- Shor 演算法
- 結語

大綱

- **基本概念**
- 量子傅立葉變換
- 量子相位估測
- Shor 演算法
- 結語

基本概念 (1)

- 本章介紹秀爾演算法（Shor's algorithm or Shor algorithm），這是一個解決大整數質因數分解（prime factorization）問題的量子演算法，具有多項式量級的時間複雜度。因為目前最快速的古典大整數質因數分解演算法仍然需要指數量級的時間複雜度，因此，秀爾演算法相較於最快速的古典演算法具有**指數量級的加速**。
- 當今使用最廣泛的 RSA 密碼系統，使用相當大的**半質數**（semiprime），也就是一個由**兩個質數相乘**而得的整數作為密碼系統的基礎。只要給定的半質數足夠大，例如，採用以 2048 位元表示的 308 位 10 進位半質數，就可以使得目前效能最好的古典演算法即使在最速的電腦上，都無法在有限時間內將給定的半質數正確分解為兩個質數，因此保證 RSA 密碼系統可以安全運作。

基本概念 (2)

- 然而，只要量子電腦的**位元數夠多**而且**錯誤率夠低**，則秀爾演算法可以在極短的時間內（例如幾分鐘之內），就完成大整數因數分解而破解 RSA 密碼系統，因而影響人們的日常生活。所以，秀爾演算法的出現，讓人們意識到量子電腦可以改變人類生活方式的強大計算能力，可說是劃時代的演算法。
- 秀爾演算法先透過古典計算方式將大整數質因數分解問題變轉為（reduce to）**模冪（modular exponentiation）函數的週期尋找（period finding）問題**，並以量子計算方式找出模冪函數週期，最後再透過古典計算方式針對找出的週期進行檢查並計算出大整數的質因數。

基本概念 (3)

- 因為秀爾演算法使用到**量子傅立葉變換**及**量子相位估測**的概念，因此本章先介紹這兩個概念。在這之後才整體說明秀爾演算法，詳細描述如何透過古典計算方式將大整數質因數分解問題變轉為模冪函數週期尋找問題，以及如何透過**量子模指數么正變換**、**量子相位估測**以及**逆量子傅立葉變換**找出模冪函數週期，最後再說明如何藉由找出的週期求得大整數的質因數，因而可以破解 RSA 密碼系統。

大綱

- 基本概念
- **量子傅立葉變換**
- 量子相位估測
- Shor 演算法
- 結語

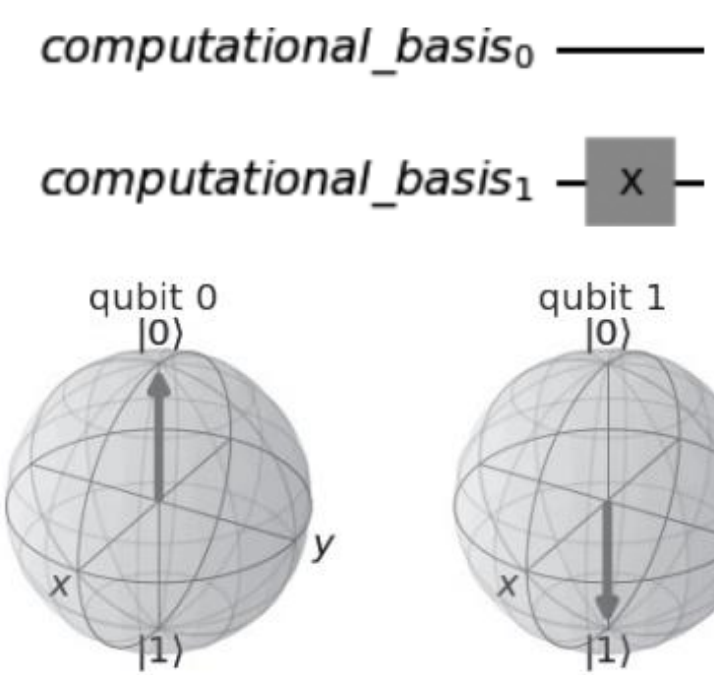
量子傅立葉變換

- 量子傅立葉變換（quantum Fourier transform, QFT）就是一種由**計算基底（computational basis）**轉換到**傅立葉基底（Fourier basis）**的變換，其中計算基底為 {|0⟩, |1⟩}，而傅立葉基底為 {|0˜⟩, |1˜⟩}={|+⟩, |−⟩}。這表示使用 H 閘就可以完成量子傅立葉變換的基底轉換，可以將使用 Z 與 -Z 的計算基底，轉換為使用 X 與 -X 的傅立葉基底。
- 以下的範例程式透過布洛赫球面展示計算基底 {|0⟩, |1⟩} 與量子傅立葉變換中使用的傅立葉基底 {|0˜⟩, |1˜⟩}={|+⟩, |−⟩}：

Program 7.1 程式解說[1]:

```
1 #Program 7.1 Show Bloch sphere for computational basis and Fourier basis
 2 from qiskit import QuantumRegister, QuantumCircuit
 3 from qiskit.quantum_info import Statevector
 4 print('='*60,'\nBelow are computational bases:')
 5 cb = QuantumRegister(2,'computational_basis')
 6 qc1 = QuantumCircuit(cb)
 7 qc1.x(1)
 8 display(qc1.draw('mpl'))
 9 state1 = Statevector.from_instruction(qc1)
10 display(state1.draw('bloch'))
```

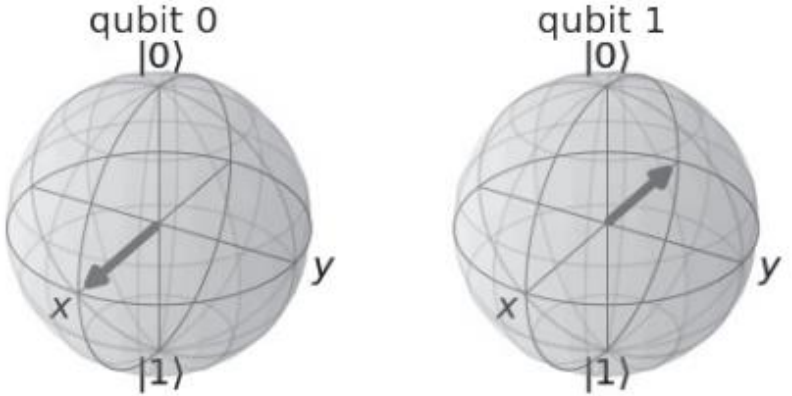
```
Below are computational bases:
```



Program 7.1 程式解說[2]:

```
11 print('='*60,'\nBelow are Fourier bases:')
12 fb = QuantumRegister(2,'fourier_basis')
13 qc2 = QuantumCircuit(fb)
14 qc2.x(1)
15 qc2.h([0,1])
16 display(qc2.draw('mpl'))
17 state2 = Statevector.from_instruction(qc2)
18 display(state2.draw('bloch'))
```

```
Below are Fourier bases:
```



量子傅立葉變換

- 實際上，使用 H 閘就可以完成 1 量子位元傅立葉變換。但是，n 量子位元傅立葉變換則複雜許多，除了使用 H 閘之外，還使用受控相位 CP 閘（controlled phase gate）來實現，以 CP 閘控制針對 Z 軸的旋轉，也就是進行相位改變的控制。
- 以下說明 n 量子位元傅立葉變換。針對 n 量子位元量子態 $|\psi\rangle$，量子傅立葉變換QFT 可以將以計算基底 {|0⟩, |1⟩} 表達的量子態 $|\psi\rangle$，變換為以傅立葉基底{|0˜⟩, |1˜⟩}={|+⟩, |−⟩} 表達的量子態 $|\psi\tilde{}\rangle$。如下所示：

$$|\tilde{\psi}\rangle = QFT|\psi\rangle$$

量子傅立葉變換

- 令 $|\psi\rangle$ 是一個表示 n 量子位元的狀態向量，並令 $N = 2^n$ 代表所有量子位元能表示的可能的量子狀態總數。若採十進位表示，則 $|\psi\rangle$ 可以表達如下：

$$|\psi\rangle = \sum_{j=0}^{N-1} a_j |j\rangle = \begin{pmatrix} a_0 \\ \vdots \\ a_{N-1} \end{pmatrix}$$，其中 $a_0, \ldots, a_{N-1}$ 為複數。

與離散傅立葉變換類似，針對量子傅立葉變換可得：

$$|\tilde{\psi}\rangle = QFT|\psi\rangle = \sum_{k=0}^{N-1} b_k |k\rangle \qquad b_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} a_j \, e^{2\pi i j k / N}$$

量子傅立葉變換

- 例如，考慮 $n = 1, N = 2^n = 2$，也就是考慮 1 量子位元狀態 $|\psi\rangle$。若採十進位表示，則可以將 $|\psi\rangle$描述為：

$$|\psi\rangle = a_0|0\rangle + a_1|1\rangle$$

依照 QFT 轉換式可得：

$$b_k = \frac{1}{\sqrt{2}} \sum_{j=0}^{1} a_j \, e^{2\pi i j k/2}, k = 0, 1$$

也就是：

$$b_0 = \frac{1}{\sqrt{2}} \sum_{j=0}^{1} a_j = \frac{1}{\sqrt{2}}(a_0 + a_1)$$

$$b_1 = \frac{1}{\sqrt{2}} \sum_{j=0}^{1} a_j \, e^{2\pi i j/2} = \frac{1}{\sqrt{2}}(a_0 + a_1 e^{i\pi}) = \frac{1}{\sqrt{2}}(a_0 - a_1)$$

量子傅立葉變換

- 因此，可以得到以下的式子：

$$QFT|\psi\rangle = QFT(a_0|0\rangle + a_1|1\rangle) = \frac{1}{\sqrt{2}}(a_0 + a_1)|0\rangle + \frac{1}{\sqrt{2}}(a_0 - a_1)|1\rangle$$

- 如下式所示，1 量子位元狀態 $|\psi\rangle$ 的 QFT 變換可以透過 H 閘的操作完成：

$$QFT|\psi\rangle = H|\psi\rangle = H(a_0|0\rangle + a_1|1\rangle) = \frac{1}{\sqrt{2}}(a_0 + a_1)|0\rangle + \frac{1}{\sqrt{2}}(a_0 - a_1)|1\rangle$$

量子傅立葉變換

- 以下再舉一個量子傅立葉變換的例子，考慮 $n = 2, N = 2^n = 4$，也就是考慮 2 量子位元狀態 $|\psi\rangle$，若採十進位表示，則可以將 $|\psi\rangle$ 描述為：

$$|\psi\rangle = a_0|0\rangle + a_1|1\rangle + a_2|2\rangle + a_3|3\rangle$$

依照 QFT 變換式可得：

$$b_k = \frac{1}{2}\sum_{j=0}^{3} a_j \, e^{2\pi i j k/4}, k = 0, 1, 2, 3$$

$$b_0 = \frac{1}{2}\sum_{j=0}^{3} a_j = \frac{1}{2}(a_0 + a_1 + a_2 + a_3)$$

$$b_1 = \frac{1}{2}\sum_{j=0}^{3} a_j \, e^{2\pi i j/4} = \frac{1}{2}(a_0 + a_1 e^{i\pi/2} + a_2 e^{i\pi} + a_3 e^{3i\pi/2})$$

$$b_2 = \frac{1}{2}\sum_{j=0}^{3} a_j \, e^{4\pi i j/4} = \frac{1}{2}(a_0 + a_1 e^{i\pi} + a_2 e^{2i\pi} + a_3 e^{3i\pi})$$

$$b_3 = \frac{1}{2}\sum_{j=0}^{3} a_j \, e^{6\pi i j/4} = \frac{1}{2}(a_0 + a_1 e^{3i\pi/2} + a_2 e^{3i\pi} + a_3 e^{9i\pi/2})$$

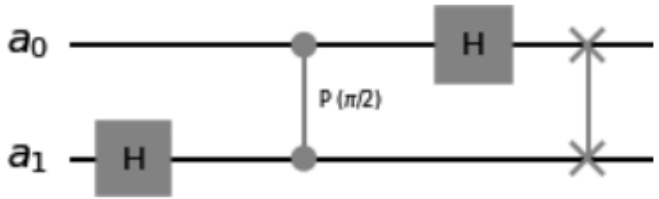
量子傅立葉變換

- 與 1 量子位元狀態 $|\psi\rangle$ 的 QFT 變換不同的是，2 量子位元狀態或更多量子位元狀態的 QFT 變換除了使用 H 閘之外，還需要使用受控相位閘操作。
- 以下的範例程式展示可以執行 2 位元量子傅立葉變換的量子線路：

Program 7.2 程式解說:

```
1 #Program 7.2 Build 2-qubit QFT quantum circuit
2 from qiskit import QuantumRegister, QuantumCircuit
3 from math import pi
4 ar = QuantumRegister(2,'a')
5 qc = QuantumCircuit(ar)
6 qc.h(1)
7 qc.cp(pi/2, 0, 1)
8 qc.h(0)
9 qc.swap(0,1)
10 print('Below is the quantum Fourier transform (QFT) circuit:')
11 display(qc.draw('mpl'))
```

```
Below is the quantum Fourier transform (QFT) circuit:
```



量子傅立葉變換

- 上列的範例程式展示 2 位元量子傅立葉變換的量子線路，其做法為首先在索引值為 1 的最高有效量子位元加上 H 閘，然後加上受控相位 CP 閘，控制索引值為 1 的最高有效量子位元針對 Z 軸進行或不進行旋轉，緊接著再於索引值為 0 的量子位元加上 H 閘。2 位元量子傅立葉變換使用的 CP 閘的控制位元為索引值為 0 的量子位元，目標位元則為索引值為 1 的量子位元，而其相位旋轉弧度為 π/2。因為這個量子線路產生出來的量子位元狀態與量子傅立葉變換產生的量子位元狀態的先後順序恰好相反，因此量子線路的最後需要使用 SWAP 閘對調量子位元的順序。

量子傅立葉變換

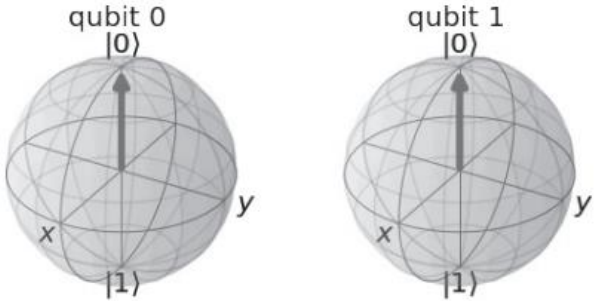
- 以下的範例程式設定不同的 2 量子位元初始狀態，包括狀態 |00⟩、|01⟩、|10⟩ 與 |11⟩。範例程式先顯示量子位元的初始狀態向量及其布洛赫球面，然後再透過 QFT的線路進行量子傅立葉變換，最後顯示經過 QFT 變換後的量子位元狀態向量及其布洛赫球面。

Program 7.3 程式解說[1]:

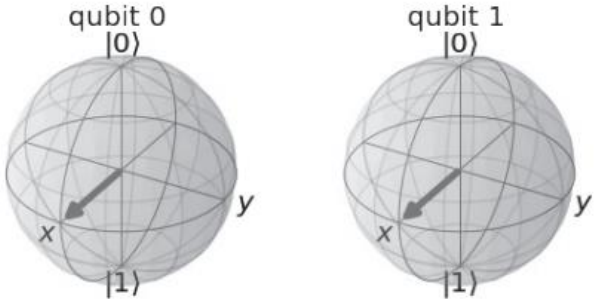
```
1 #Program 7.3 Apply QFT to qubit with various initial state
 2 from qiskit import QuantumRegister, QuantumCircuit
 3 from qiskit.quantum_info import Statevector
 4 from qiskit.visualization import array_to_latex
 5 from math import pi
 6 two_bits = ['00','01','10','11']
 7 for bits in two_bits:
 8             ar = QuantumRegister(2,'a')
 9             qc = QuantumCircuit(ar)
10             qc.initialize(bits,ar)
11             state1 = Statevector.from_instruction(qc)
12             print('='*75,'\nBelow is for qubits: q0 =',bits[0],'; q1 =',bits[1])
13             display(array_to_latex(state1, prefix='\\text{Statevector before
14             display(state1.draw('bloch'))
```

```
Below is for qubits: q0 = 0 ; q1 = 0
```

Statevector before QFT: $\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$



Statevector after QFT: $\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$

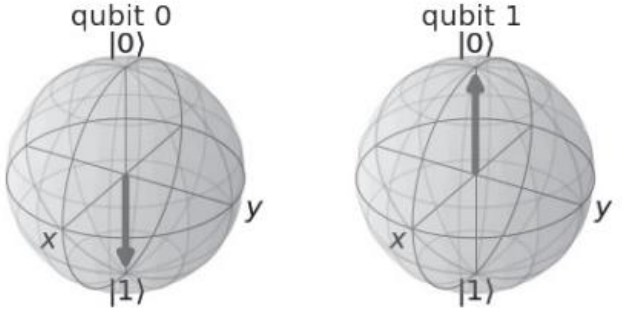


Program 7.3 程式解說[2]:

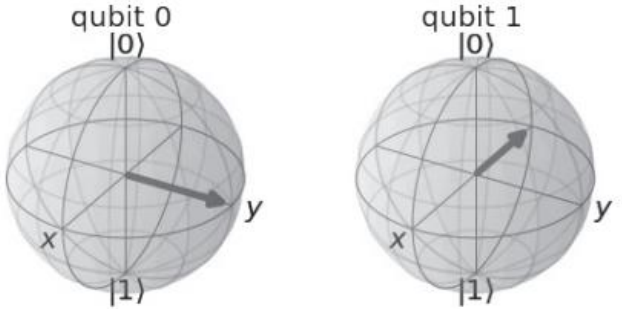
```
15          qc.h(1)
16          qc.cp(pi/2, 0, 1)
17          qc.h(0)
18          qc.swap(0,1)
19          state2 = Statevector.from_instruction(qc)
20          display(array_to_latex(state2, prefix='\\text{Statevector after
21          display(state2.draw('bloch'))
```

```
Below is for qubits: q0 = 0 ; q1 = 1
```

Statevector before QFT: $\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$



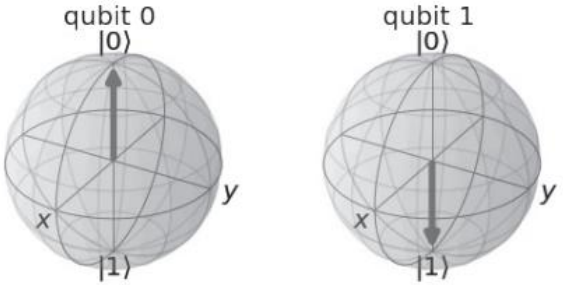
Statevector after QFT: $\begin{bmatrix} \frac{1}{2} & \frac{1}{2}i & -\frac{1}{2} & -\frac{1}{2}i \end{bmatrix}$



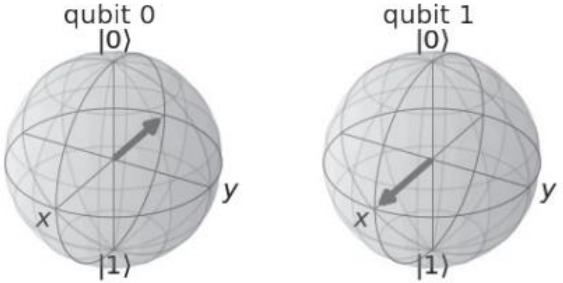
Program 7.3 程式解說[3]:

```
Below is for qubits: q0 = 1 ; q1 = 0
```

Statevector before QFT: $\begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}$

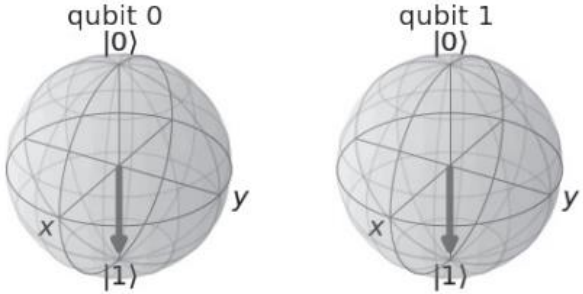


Statevector after QFT: $\begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \end{bmatrix}$

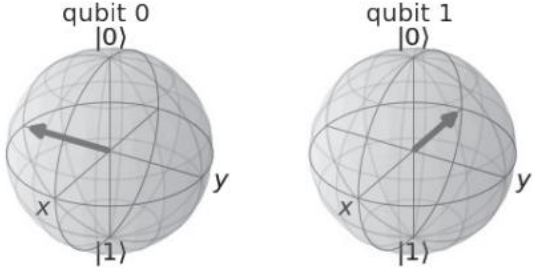



```
Below is for qubits: q0 = 1 ; q1 = 1
```

Statevector before QFT: $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$



Statevector after QFT: $\begin{bmatrix} \frac{1}{2} & -\frac{1}{2}i & -\frac{1}{2} & \frac{1}{2}i \end{bmatrix}$



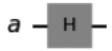
量子傅立葉變換

- 以下的範例程式展示 n 個量子位元的一般化量子傅立葉變換（QFT）量子線路，範例程式中將 n 設定為 1、2、3、4 以顯示對應的 QFT 量子線路：

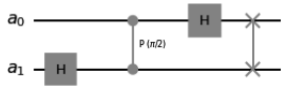
Program 7.4 程式解說:

```
1 #Program 7.4 Define funciton to build n-qubit QFT quantum circuit
 2 from qiskit import QuantumRegister, QuantumCircuit
 3 from math import pi
 4 def qft(n):
 5             ar = QuantumRegister(n,'a')
 6             qc = QuantumCircuit(ar)
 7             for hbit in range(n-1,-1,-1):
 8                         qc.h(hbit)
 9                         for cbit in range(hbit):
10                                     qc.cp(pi/2**(hbit-cl
11             for bit in range(n//2):
12                         qc.swap(bit,n-bit-1)
13             return qc
14 for i in range(1,5):
15             print('Below is the QFT circuit of',i,'qubit(s):'
16             display(qft(i).draw('mpl'))
```

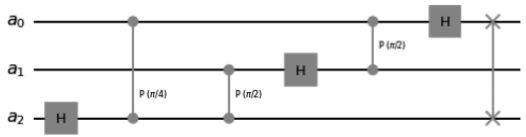
```
Below is the QFT circuit of 1 qubit(s):
```



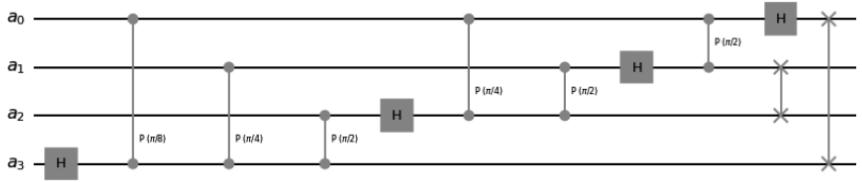

```
Below is the QFT circuit of 2 qubit(s):
```




```
Below is the QFT circuit of 3 qubit(s):
```




```
Below is the QFT circuit of 4 qubit(s):
```



量子傅立葉變換

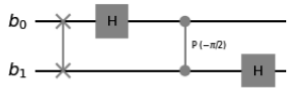
- 將量子傅立葉變換的量子線路逆向就得到逆量子傅立葉變換（inverse quantum Fourier transform, IQFT）的量子線路。以下的範例程式展示 n 個量子位元的一般逆量子傅立葉變換（IQFT）量子線路，其中 n 先設定為 1、2、3、4，以作為範例來顯示 IQFT 量子線路：

Program 7.5 程式解說:

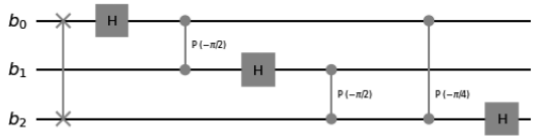
```
1 #Program 7.5 Define function to build n-qubit IQFT quantum circuit
 2 from qiskit import QuantumRegister, QuantumCircuit
 3 from math import pi
 4 def iqft(n):
 5             br = QuantumRegister(n,'b')
 6             qc = QuantumCircuit(br)
 7             for sbit in range(n//2): #sbit: for swap qubit
 8                         qc.swap(sbit,n-sbit-1)
 9             for hbit in range(0,n,1): #hbit: for h-gate qubi
10                         for cbit in range(hbit-1,-1,-1): #c
11                                     qc.cp(-pi/2**(hbit-cb
12                          qc.h(hbit)
13             return qc
14 for i in range(1,5):
15             print('Below is the IQFT circuit of',i,'qubit(s):')
16             display(iqft(i).draw('mpl'))
```

```
Below is the IQFT circuit of 1 qubit(s):
```

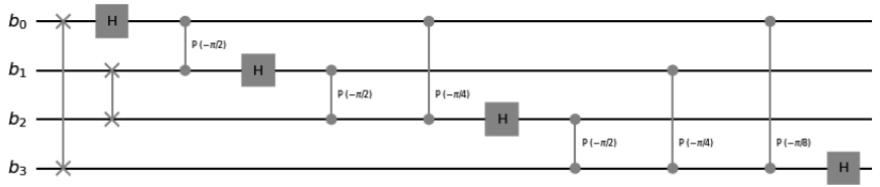

```
Below is the IQFT circuit of 2 qubit(s):
```




```
Below is the IQFT circuit of 3 qubit(s):
```




```
Below is the IQFT circuit of 4 qubit(s):
```

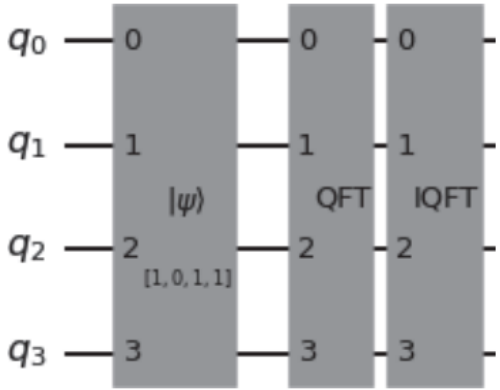


量子傅立葉變換

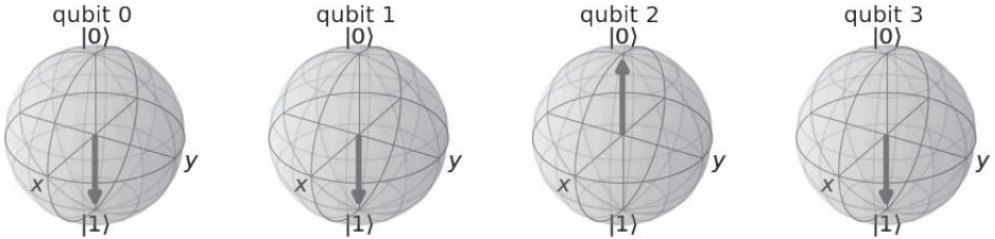
- 到目前為止已經介紹如何設計程式，實現量子傅立葉變換量子線路及逆量子傅立葉變換量子線路。以下的範例程式展示如何將 4 個量子位元 q_3、q_2、q_1、q_0 的初始值設定為 '1011'，然後經過量子傅立葉變換量子線路之後以布洛赫球面顯示量子位元狀態，再經過逆量子傅立葉變換量子線路之後再以布洛赫球面顯示量子位元狀態，因而得以可以看出量子位元狀態又回復原來的 '1011' 狀態了。請注意，範例程式是以 q_0、q_1、q_2、q_3 的順序顯示量子位元對應的布洛赫球面，這與字串 '1011' 的最左邊字元用於表達最高有效位元 q_3，而最右邊字元用於表達最低有效位元 q_0 的順序相反。

Program 7.6 程式解說[1]:

```
1 #Program 7.6 Apply QFT and then IQFT to qubit
 2 from qiskit import QuantumCircuit
 3 from qiskit.quantum_info import Statevector
 4 qc = QuantumCircuit(4)
 5 qc.initialize('1011',range(4))
 6 state0 = Statevector.from_instruction(qc)
 7 qc.append(qft(4).to_gate(label='QFT'),range(4))
 8 state1 = Statevector.from_instruction(qc)
 9 qc.append(iqft(4).to_gate(label='IQFT'),range(4))
10 state2 = Statevector.from_instruction(qc)
11 display(qc.draw('mpl'))
12 print('Statevector before QFT:')
13 display(state0.draw('bloch'))
14 print('Statevector after QFT:')
```



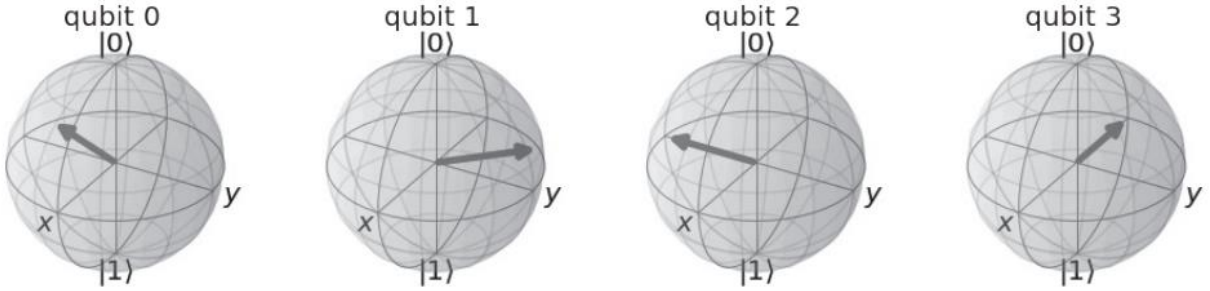

```
Statevector before QFT:
```



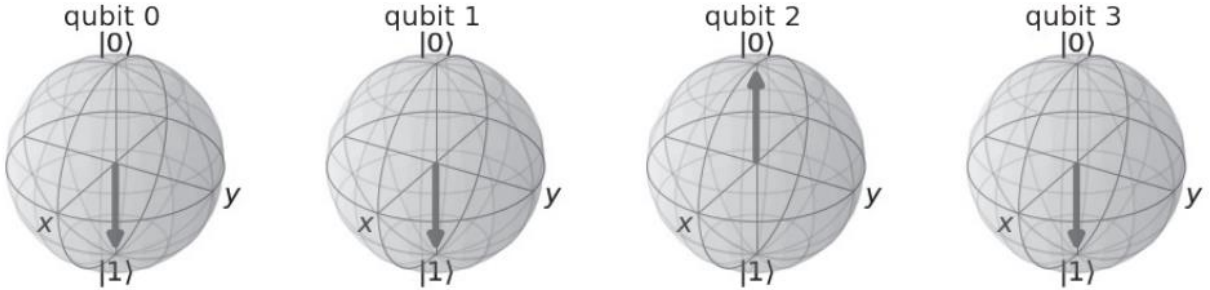
Program 7.6 程式解說[2]:

```
15 display(state1.draw('bloch'))
16 print('Statevector after IQFT:')
17 display(state2.draw('bloch'))
```

```
Statevector after QFT:
```




```
Statevector after IQFT:
```



大綱

- 基本概念
- 量子傅立葉變換
- **量子相位估測**
- Shor 演算法
- 結語

量子相位估測

- 量子相位估測（quantum phase estimation, QPE）演算法也稱為量子本徵值估測（quantum eigenvalue estimation）演算法，其目的如下所列：
- 給定一個么正變換 U，滿足 $U|\psi\rangle = e^{2\pi i\lambda}|\psi\rangle$，其中 $|\psi\rangle$ 是 U 的本徵態（eigenstate），$e^{2\pi i\lambda}$ 是對應的具有相位（phase）λ 的本徵值（eigenvalue）。QPE 演算法的目的是找出給定么正變換 U 本徵值的相位 λ。

量子相位估測

- QPE 演算法使用 n 個量子計數位元紀錄么正變換 U 本徵值的相位 λ 乘上 2^n 的值（也就是 $\lambda 2^n$ ），並以傅立葉基底的方式儲存，然後再透過逆量子傅立葉變換（inverse quantum Fourier transform）變換回以計算基底表示的對應值（或狀態）以便進行測量。最後，只要測量出計數位元的值，將這個值除以 2^n 就可以計算出相位 λ。
- 以下的範例程式使用 n (n = 2) 個量子計數位元紀錄么正變換 S 閘本徵值的相位 λ乘上 2^n (2^n = 4) 的值，然後經過逆量子傅立葉變換後測量出計數位元的值。

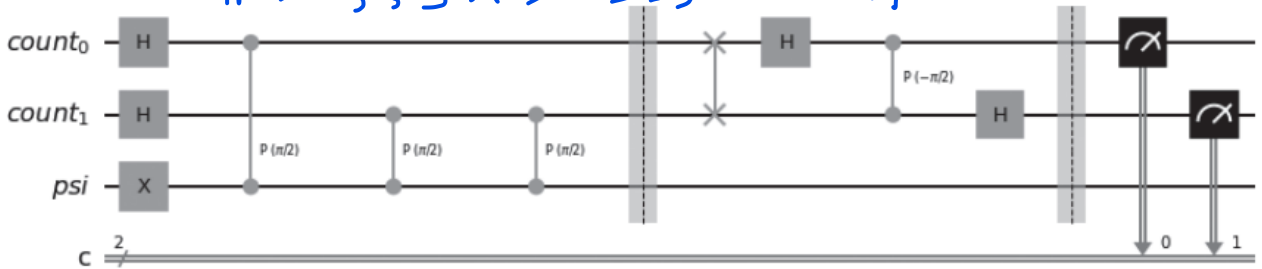
Program 7.7 程式解說[1]:

```
1 #Program 7.7 Use QPE to estimate phase of S-gate
 2 from qiskit import QuantumRegister, QuantumCircuit, ClassicalRegister,execute
 3 from qiskit.providers.aer import AerSimulator
 4 from qiskit.visualization import plot_histogram
 5 from math import pi
 6 count_no = 2 #the number of count qubits
 7 countreg = QuantumRegister(count_no,'count')
 8 psireg = QuantumRegister(1,'psi')
 9 creg = ClassicalRegister(count_no,'c')
10 qc = QuantumCircuit(countreg,psireg,creg)
11 for countbit in range(count_no):
12             qc.h(countbit)
13 qc.x(psireg)
14 repeat = 1
```

$e^{ix} = \cos x + i \sin x$

$U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & e \\ 0 & 0 & 0 & i \end{bmatrix}$ $P(\frac{\pi}{2}) = S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$

$|psi\rangle = SSS \times |0\rangle = SSS\,|1\rangle = \begin{bmatrix} 0 \\ -i \end{bmatrix} = -i\begin{bmatrix} 0 \\ 1 \end{bmatrix}$

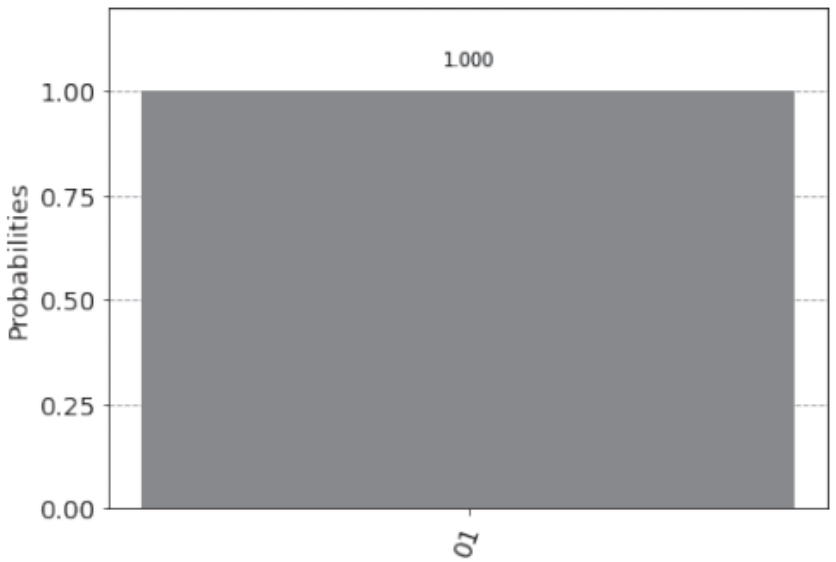


Program 7.7 程式解說[2]:

```
15 for countbit in range(count_no):
16             for r in range(repeat):
17                         qc.cp(pi/2,countbit,psireg)
18             repeat *= 2
19 qc.barrier()
20 for sbit in range(count_no//2): #sbit: for swap qubit
21             qc.swap(sbit,count_no-sbit-1)
22 for hbit in range(0,count_no,1): #hbit: for h-gate qubit
23             for cbit in range(hbit-1,-1,-1): #cbit: for count
24                         qc.cp(-pi/2**(hbit-cbit), cbit, hbit)
25             qc.h(hbit)
26 qc.barrier()
```

```
Total counts for qubit states are: {'01': 1000}

Out[11]:
```



Program 7.7 程式解說[3]:

```
27 qc.measure(range(count_no),range(count_no))
28 display(qc.draw('mpl'))
29 sim = AerSimulator()
30 job=execute(qc, backend=sim, shots=1000)
31 result = job.result()
32 counts = result.get_counts(qc)
33 print("Total counts for qubit states are:",counts)
34 plot_histogram(counts)
```

量子相位估測

- 上述範例程式藉由帶 $\pi/2$ 參數的 CP 閘實現 CS 閘，針對么正變換 S 閘進行量子相位估測，得到的測量結果為 '01'，代表的數值為 1，將這個值除以 2^n（n 是量子計數位元的個數），就可以求出相位了。因為範例程式採用的量子計數位元個數為 2，因此由測量結果 '01' 所代表的數值為 1 可以推導出相位為 $1/2^n = 1/2^2 = 1/4 = 0.25$。這代表 S 閘操作會使得原來的量子位元狀態產生 1/4（週期）的相位改變，這相當於針對 Z 軸旋轉$2\pi / 4 = \pi / 2$ 弧度的相位改變，確實符合 CS 閘操作的特性。

量子相位估測

- 以下說明一般化量子相位估測 QPE 量子線路的設計方式。QPE 透過相位回擊將么正變換 U 的本徵值 $e^{2\pi i\lambda}$ 對應的相位 λ 以傅立葉基底寫入 n 個量子計數位元中；然後再使用逆量子傅立葉變換，將量子計數位元變換為以計算基底表示以對其進行測量，最後求出本徵值 $e^{2\pi i\lambda}$ 對應的相位 λ。
- 具體的說，當使用量子計數位元來控制么正變換 U 而形成對應的受控 CU 閘時，若控制量子位元（也就是計數位元）處於 $|+\rangle$ 狀態時將產生相位回擊的作用，此時控制量子位元位的相位將旋轉與本徵值 $e^{2\pi i\lambda}$ 對應的 λ 弧度。

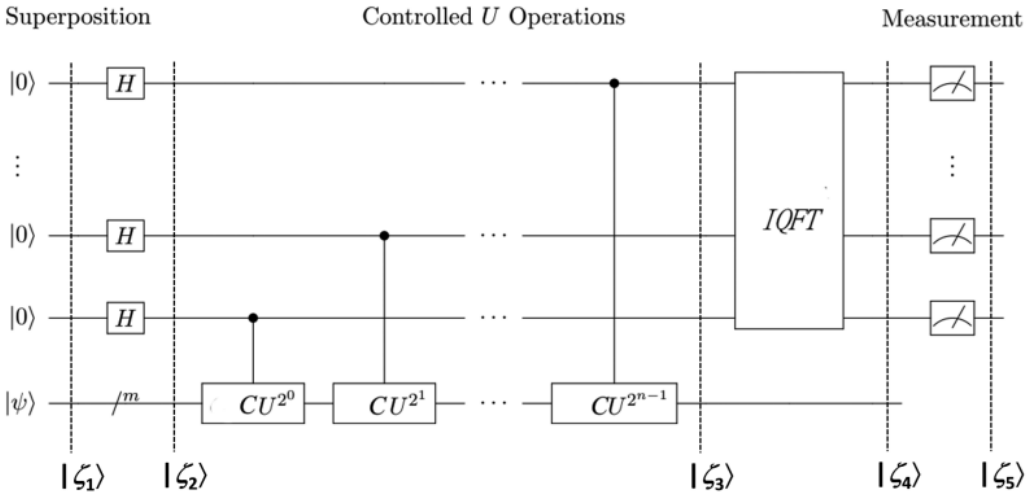
量子相位估測

- 因此，連續進行 *CU* 閘 k 次，則控制量子位元位的相位將旋轉 $k\lambda$ 弧度。QPE 將執行 2^0、2^1、...、2^n 次*CU* 閘操作視為單一的 CU^{2^0} 閘、CU^{2^1} 閘、...、CU^{2^n} 閘，分別以不同的計數位元為控制位元，可以將相位 λ 編碼為傅立葉基底中 0 到 2^n 之間的數值儲存於計數位元中。在這之後，再透過逆量子傅立葉變換就可以將量子計數位元變換為以計算基底表示，以對其進行測量並計算出正確的相位 λ。
- 請注意，一個 U^{2^n} 么正變換可以透過平方求冪（exponentiation by squaring）的方式，以 n 次計算獲得等價的變換。平方求冪的概念描述如下。對一個么正變換 *U* 及一個正整數 k，U^k 可以依照以下的方式計算：

$$U^k = \begin{cases} U\,(U^2)^{\frac{k-1}{2}}, & \text{if } k \text{ is odd} \\ (U^2)^{\frac{k}{2}}, & \text{if } k \text{ is even.} \end{cases}$$

量子相位估測

- 利用平方求冪的概念只需要 「$\log_2 k$ ⌉次計算就可以求得 U^k。依照以上的推論，若令 $k = 2^j$，則需要 j 次么正矩陣乘法計算就可以求得 U^{2^j} ，其中 $j = 0, 1,\ldots,n$。



量子相位估測

以下說明一般化量子相位估測的執行步驟：

1. 準備量子位元：
   準備 m 個量子位元儲存本徵態 $|\psi\rangle$，並準備 n 個量子計數位元儲存本徵值對應的相位 λ 乘上 2^n 的值，也就是 $2^n\lambda$。令這個階段的整體量子位元狀態為 $|\zeta_1\rangle$，可以推得： $|\zeta_1\rangle = |0\rangle^{\otimes n}|\psi\rangle$
1. 準備量子疊加態：
   所有 n 個量子計數位元都同時經過 H 閘的操作，以形成 $|+\rangle^{\otimes n}$ 狀態。 令這個階段的整體量子位元狀態為 $|\zeta_2\rangle$，可以推得： $|\zeta_2\rangle = |+\rangle^{\otimes n}|\psi\rangle = \frac{1}{\sqrt{2^n}}(|0\rangle + |1\rangle)^{\otimes n}|\psi\rangle$
1. 進行受控 *CU* 閘操作：
   受控么正變換 *CU* 在控制位元為 $|1\rangle$ 的時候，m 個狀態位元會進行 *U* 閘操作。

量子相位估測

因為 U 是一個具有本徵態 $|\psi\rangle$ 及本徵值 $e^{2\pi i\lambda}$ 的么正變換，也就是

$$U|\psi\rangle = e^{2\pi i\lambda}|\psi\rangle$$

因此可以推得：

$$U^{2^j}|\psi\rangle = U^{2^j-1}U|\psi\rangle = U^{2^j-1}e^{2\pi i\lambda}|\psi\rangle = \cdots = e^{2\pi i 2^j\lambda}|\psi\rangle$$

令運作所有的 CU^{2^j} 閘， $0 \le j \le n-1$，之後整體量子位元的狀態為 $|\zeta_3\rangle$，引用以下的關係 $|0\rangle\otimes|\psi\rangle+|1\rangle \otimes e^{2\pi i\lambda}|\psi\rangle = (|0\rangle + e^{2\pi i\lambda}|1\rangle) \otimes |\psi\rangle$ 可以推得：

$$|\zeta_3\rangle = \frac{1}{\sqrt{2^n}}\left(|0\rangle + e^{2\pi i\lambda 2^{n-1}}|1\rangle\right)\otimes\cdots\otimes\left(|0\rangle + e^{2\pi i\lambda 2^1}|1\rangle\right)\otimes\left(|0\rangle + e^{2\pi i\lambda 2^0}|1\rangle\right)\otimes|\psi\rangle$$

$$= \frac{1}{\sqrt{2^n}}\sum_{u=0}^{2^n-1} e^{2\pi i\lambda u}|u\rangle\otimes|\psi\rangle$$

u 代表 n 個量子位元能夠代表由 0 到 2^n-1 的整數。

量子相位估測

4. 逆量子傅立葉變換：
   讀者應該已經注意到上列 $|\zeta_3\rangle$ 的形式與量子計數位元經過量子傅立葉變換的形式相同，也就是說，傅立葉變換將計數位元由以計算基底表示變換為以傅立葉基底表示。因此，只要再透過逆量子傅立葉變換就可以將計數位元由傅立葉基底表示轉回由計算基底表示，如此就可以進行後續的測量。
   n 位元的量子傅立葉變換表示如下：

$$QFT|x\rangle :$$

$$= \frac{1}{\sqrt{2^n}} \left(|0\rangle + e^{\frac{2\pi i}{2}x}|1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^2}x}|1\rangle\right) \otimes \ldots \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^{n-1}}x}|1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^n}x}|1\rangle\right)$$

量子相位估測

將上式中的 x 取代為 $2^n\lambda$ 再加上與 $|\psi\rangle$ 進行張量積運算就可以得到與 $|\zeta_3\rangle$ 完全相同的結果。因此，針對計數位元進行逆量子傅立葉變換就可以求得 $|2^n\lambda\rangle$。令進行逆量子傅立葉變換之後的整體量子位元狀態為 $|\zeta_4\rangle$，可以推得：

$$|\zeta_4\rangle = \frac{1}{\sqrt{2^n}} \sum_{u=0}^{2^n-1} e^{2\pi i \lambda u} |u\rangle \otimes |\psi\rangle \xrightarrow{IQFT} \frac{1}{\sqrt{2^n}} \sum_{v=0}^{2^n-1} \sum_{u=0}^{2^n-1} e^{-\frac{2\pi i u}{2^n}(v-2^n\lambda)} |v\rangle \otimes |\psi\rangle$$

量子相位估測

5. 測量與計算
   在$|\zeta_4\rangle$ 之中，在 $v = 2^n\lambda$ 的 機率振幅最大。因此，若 $2^n\lambda$ 是一個整數，則這個整數在測量時出現的機率會最高。令測量之後的計數位元狀態為 $|\zeta_5\rangle$，可以推得：$|\zeta_5\rangle = |2^n\lambda\rangle$

- Nielsen 及 Chuang 兩人在著名的「Quantum Computation and Quantum Information」一書（2011 年第 10 版）中，針對 $2^n\lambda$ 不是整數的情況進行探討，書中提到測量結果在接近 $x = 2^n\lambda$ 附近出現峰值的機率高於 $4/\pi^2 \approx 40\%$

大綱

- 基本概念
- 量子傅立葉變換
- 量子相位估測
- **Shor 演算法**
- 結語

Shor 演算法

- 給定大整數 $N = p \times q$，其中 p 與 q 是兩個大於 2 的不同質數，秀爾（Shor）演算法可以將整數 N 因數分解為 p 與 q 的乘積。秀爾演算法利用歐拉定理（Euler's theorem）的概念，也就是若 a 與 N 互質，則模冪函數 $f(x) = a^x \pmod{N}$ 是週期函數。具體的說，秀爾演算法先使用古典演算法找出對應大整數 N 因數分解的模冪函 $f(x)$，然後使用量子計算方式快速找到 $f(x)$ 模冪函數的週期，因而能夠以多項式時間複雜度將大整數 N 因數分解為兩個質數的乘積。

Shor 演算法

- 以下說明秀爾演算法的執行步驟：
  - 步驟 1：在 2 和 $N - 1$ 之間隨機選擇一個整數 a。
  - 步驟 2：使用歐幾里德古典演算法計算 $g = gcd(a, N)$。
  - 步驟 3：若 $g \neq 1$，則 g 是 N 的因數，回傳 $p = g$ 及 $q = N/g$，然後結束演算法。
  - 步驟 4：以量子計算方式找出 $f(x) = a^x \pmod N$ 的週期 r，$r > 1$。其中 r 是使得 $f(x) = f(x + r)$ 的最小整數。也就是說，r 是使得 $a^r = 1 \pmod N$ 的最小整數，$r > 1$。
  - 步驟 5：若 r 是奇數，則跳至步驟 1。

Shor 演算法

- 以下說明秀爾演算法的執行步驟：
  - 步驟 6：若 r 是偶數，則計算 $a^{r/2} \pmod N$。
  - 步驟 7：使若 $a^{r/2} = -1 \pmod N$，則跳至步驟 1。
  - 步驟 8：$gcd\,(a^{r/2} + 1, N)$ 及 $gcd\,(a^{r/2} - 1, N)$ 之間必然存在一個 N 的非必然（non-trivial）因數。令 p 為這個因數，並令 $q = N/p$，回傳 p 與 q 為 N 的因數，然後結束演算法。

Shor 演算法

由秀爾演算法的步驟 4 得知：

$$a^r - 1 = 0 \pmod{N}$$

又由步驟 6 得知 r 為偶數，因此可以推導得出：

$$(a^{r/2} + 1)(a^{r/2} - 1) = 0 \pmod{N}$$

因為根據步驟 4，r 是使得 $a^r = 1 \pmod{N}$ 的最小整數，因此可得：

$a^{r/2} - 1 \neq 0 \pmod{N}$，也就是 $a^{r/2} - 1$ 不是 N 的倍數。

又由步驟 7 得知：

$a^{r/2} + 1 \neq 0 \pmod{N}$，也就是 $a^{r/2} + 1$ 不是 N 的倍數。

Shor 演算法

- 因此可以推導出：
  $(a^{r/2} + 1)(a^{r/2} - 1)$ 必定是 N 的倍數，也就是：
  $(a^{r/2} + 1)(a^{r/2} - 1) = kN$ ，其中 k 為整數。
- 以下用反證法的方式推導出 $a^{r/2} + 1$ 與 N 不互質或 $a^{r/2} - 1$ 與 N 不互質；也就是 $gcd\,(a^{r/2} + 1, N) \neq 1$ 或 $gcd\,(a^{r/2} - 1, N) \neq 1$。
  假設 $gcd\,(a^{r/2} + 1, N) = 1$ 且 $gcd\,(a^{r/2} - 1, N) = 1$，則根據貝祖定理（Bezout's lemma）可得：
  $$u(a^{r/2} + 1) + vN = 1$$，其中 u, v 為整數。
  將上式等號的兩邊各乘上 $(a^{r/2} - 1)$ 則可得：
  $$u(a^{r/2} + 1)(a^{r/2} - 1) + vN(a^{r/2} - 1) = (a^{r/2} - 1)$$，其中 u, v 為整數。

Shor 演算法

- 將$(a^{r/2} + 1)(a^{r/2} - 1) = kN$ 代入上式後可得：

  $ukN + vN(a^{r/2} - 1) = (a^{r/2} - 1)$

- 上式代表 $a^{r/2} - 1$ 是 N 的倍數，而這是一個矛盾條件。因此，前列的假設 $gcd\,(a^{r/2} + 1, N) = 1$ 且 $gcd\,(a^{r/2} - 1, N) = 1$ 不成立，也就是說：

  $gcd\,(a^{r/2} + 1, N) \neq 1$ 或 $gcd\,(a^{r/2} - 1, N) \neq 1$。所以，$a^{r/2} + 1$ 與 $a^{r/2} - 1$ 之中至少有一個與 N 有非顯然的（non-trivial）公因數，也就是不是 1 也不是 N 的公因數。

- 因為 N 只有兩個因數，因此可以推導出 N 的因數為：

  $p = gcd\,(a^{r/2} \pm 1, N)$ 及 $q = N/gcd\,(a^{r/2} \pm 1, N)$

Shor 演算法

- 以上的秀爾演算法的步驟中，除了步驟 4 之外，其他的步驟都可以透過古典計算方式以多項式時間複雜度完成。例如，可以使用歐幾里得演算法以 O(log min(x,y))的時間複雜度求出兩個整數 x 及 y 的最大公因數。步驟 4 的作用為達成週期尋找（period finding）功能，若使用古典計算模式來實現週期尋找功能，會引發超多項式（super-polynomial）的時間複雜度。但是若使用量子計算模式來實現週期尋找功能，則可以在多項式時間複雜度完成。
- 我們先以下列的範例程式示範如何完全以古典計算方式實現秀爾演算法，然後再以另一個範例程式示範如何使用量子計算方式實現秀爾演算法中的週期尋找功能。

Program 7.8 程式解說[1]:

```
1 #Program 7.8 Classical Shor Algorithm
 2 from random import randint
 3 from math import gcd
 4 def period_finding(a,N):
 5             for r in range(1,N):
 6                         if (a**r) % N == 1:
 7                                     return r
 8 def shor_alg(N):
 9             while True:
10                         a=randint(2,N-1)
11                         g=gcd(a,N)
12                         if g!=1:
13                                     p=g
```

Program 7.8 程式解說[2]:

```
14                      q=N//g
15                      return p,q
16          else:
17                      r=period_finding(a,N)
18                      if r % 2 != 0:
19                              continue
20                      elif a**(r//2) % N == -1 % N:
21                              continue
22                      else:
23                              p=gcd(a**(r//2)+1,N)
24                              if p==1 or p==N:
25                                      p=gcd(a**(r//2)-1,N)
26                              q=N//p
27          return p,q
28 for N in [15, 21, 35, 913, 2257, 10999]:
29          print(f'Factors of {N}: {shor_alg(N)}')
```

```
Factors of 15: (5, 3)
Factors of 21: (3, 7)
Factors of 35: (5, 7)
Factors of 913: (11, 83)
Factors of 2257: (61, 37)
Factors of 10999: (17, 647)
```

Shor 演算法

- 以下說明如何使用量子計算的方式完成秀爾演算法中的週期尋找功能。其主要的做法為使用么正變換的量子相位估測，也就是使用受控量子閘的相位回擊以及逆量子傅立葉變換等之前已經介紹過的概念。以下詳細解釋秀爾演算法如何完成量子週期尋找功能。
- 令 U 是一個么正變換，定義如下：

  $U|y\rangle \equiv |ay \bmod N\rangle$
- 先考慮一個簡單的範例，即令 $a = 7$ 與 $N = 15$。由 $y = |1\rangle$ 開始，連續進行么正變換 U 可以得到以下的計算過程：

Shor 演算法

- $U|1\rangle = |7 \cdot 1 \bmod 15\ \rangle = |7\rangle$
  $U^2|1\rangle = UU|1\rangle = U|7 \cdot 1 \bmod 15\rangle = |7^2 \cdot 1 \bmod 15\rangle = |4\rangle$
  $U^3|1\rangle = UUU|1\rangle = U|7^2 \cdot 1 \bmod 15\rangle = |7^3 \cdot 1 \bmod 15\rangle = |13\rangle$
  $U^4|1\rangle = UUUU|1\rangle = U|7^3 \cdot 1 \bmod 15\rangle = |7^4 \cdot 1 \bmod 15\rangle = |1\rangle$

.

.

.

.

Shor 演算法

- 可以發現，當連續進行 4 次么正變換 U 時，可以回到開始的 $y = |1\rangle$ 狀態，因此可以確定週期 $r = 4$。
- 因為量子狀態 $|y\rangle = |1\rangle$ 每次連續進行 $r = 4$ 次么正變換 U，就會再度出現週期性的狀態：$|1\rangle$、$|7\rangle$、$|4\rangle$、$|13\rangle$、$|1\rangle$、...。因此，由 $r = 4$ 個週期性出現的狀態混合在一起的疊加態 $|y_0\rangle = \frac{1}{\sqrt{r}}\sum_{k=0}^{r-1}|a^k \bmod N\rangle$ 是么正變換 U 的本徵狀態，而且其本徵值為 1。也就是說：

$$U|y_0\rangle = U\left(\frac{1}{\sqrt{r}}\sum_{k=0}^{r-1}|a^k \bmod N\rangle\right) = |y_0\rangle$$

Shor 演算法

- 此時，若考慮另外一個疊加態 $|y_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi ik}{r}} |a^k \bmod N\rangle$，則可以推導得知$|y_1\rangle$ 也是么正變換 U 的本徵狀態，而且其本徵值為 $e^{\frac{2\pi i}{r}}$，也就是：

$$U|y_1\rangle = U\left(\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi ik}{r}} |a^k \bmod N\rangle\right) = e^{\frac{2\pi i}{r}} |y_1\rangle$$

- 若在疊加態 $|y_1\rangle$ 的每一個分項乘上整數 s，其中 $0 \le s \le r-1$，則所形成的疊加態$|y_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi isk}{r}} |a^k \bmod N\rangle$ 也是么正變換 U 的本徵狀態，而且其本徵值為 $e^{\frac{2\pi is}{r}}$，也就是：

$$U|y_s\rangle = U\left(\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi isk}{r}} |a^k \bmod N\rangle\right) = e^{\frac{2\pi is}{r}} |y_s\rangle$$

Shor 演算法

- 我們可以驗證，若將所有對應 $s = 0, 1, \ldots, r-1$ 的本徵態 $U|y_s\rangle$ 加總起來，則除了 $s = 0$ 之外，其他所有 s 的值所對應不同的相位會互相抵減，因此可得：

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |y_s\rangle = |1\rangle$$

- 這也就是說，計算基底 $|1\rangle$ 是對應 $s = 0, 1, \ldots, r-1$ 的所有本徵態 $U|y_s\rangle$ 的線性組合量子態。因此，若針對這個線性組合量子態進行量子相位估測（QPE），則可以測量得到以下的相位角 λ：

$$\lambda = \frac{s}{r}$$

Shor 演算法

- 稍後會說明，在得知相位角 λ 之後，可以透過連分數（continued fraction）的方式求出 s 與 r，也就能夠達成週期尋找功能。
- 以下展示如何使用量子計算方式將整數 $N = 15$ 進行因數分解的過程。其做法為先選擇與 N 互質而且大於 1 且小於 N 的整數 a，以設計么正變換 U 的量子線路，其中 $a = 2, 4, 7, 8, 11, 13, 14$，$U|y\rangle \equiv |ay \bmod N\rangle$。若么正變換 U 的線路重複 x 次，就可以完成以下的計算：

$$a^x \pmod N$$

Shor 演算法

- 以下表格顯示針對 $N = 15$ 及 $a = 2, 4, 7, 8, 11, 13, 14$ 的情況下的各種計算結果：

a	a^x	$a^x \pmod N$	r	$gcd(a^{\frac{r}{2}} - 1, N)$	$gcd(a^{\frac{r}{2}} + 1, N)$
2	1, 2, 4, 8, 16	1, 2, 4, 8, 1	4	3	5
4	1, 4, 16, 64, 256	1, 4, 1, 4, 1	2	3	5
7	1, 7, 49, 343, 2401	1, 7, 4, 13, 1	4	3	5
8	1, 8, 64, 512, 4096	1, 8, 4, 2, 1	4	3	5
11	1, 11, 121, 1331, 14641	1, 11, 1, 11, 1	2	5	3
13	1, 13, 169, 2197, 28561	1, 13, 4, 7, 1	4	3	5
14	1, 14, 196, 2744, 38416	1, 14, 1, 14, 1	2	1	15

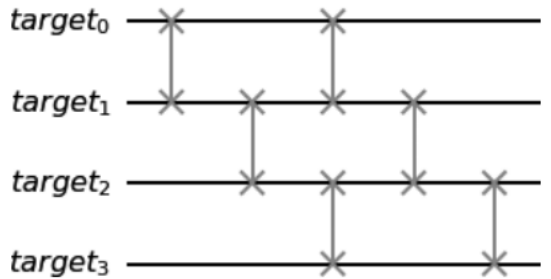
Shor 演算法

- 在以上表格中，當 a 等於 14 時，週期 r 為 2，可以推導得出 $a^{r/2} = 14^1 = -1 \pmod{N}$。
- 根據前述的秀爾演算法的步驟 7，要略過 $a = 14$ 的情況，因此以下的範例程式僅列出對應 $a = 2, 4, 7, 8, 11, 13$ 的情況設計量子線路。Monz 等人在論文「Realizationof a scalable Shor algorithm」中，針對 $N = 15$的因數分解提出使用 SWAP 閘實作么正變換 U 的量子線路，其中 $U|y\rangle \equiv |ay \bmod N\rangle$，$a = 2, 4, 7, 8, 11, 13$，若么正變換 U 的線路重複 x 次，就可以完成 $a^x \pmod{N}$ 的計算，可以參考這篇論文以獲得量子線路實作的細節及解釋。

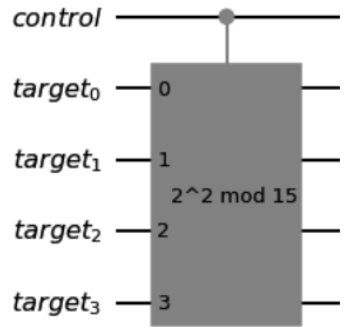
Program 7.9 程式解說[1]:

```
1 #Program 7.9 Define function to build modular exponentiation quantum circuit
 2 from qiskit import QuantumRegister, QuantumCircuit
 3 def qc_mod15(a, power, show=False):
 4         assert a in [2,4,7,8,11,13], 'Invalid value of argument a:'+s
 5         qrt = QuantumRegister(4,'target')
 6         U = QuantumCircuit(qrt)
 7         for i in range(power):
 8                 if a in [2,13]:
 9                         U.swap(0,1)
10                         U.swap(1,2)
11                         U.swap(2,3)
12                 if a in [7,8]:
13                         U.swap(2,3)
14                         U.swap(1,2)
15                         U.swap(0,1)
```

Below is the circuit of U of "2^2 mod 15":



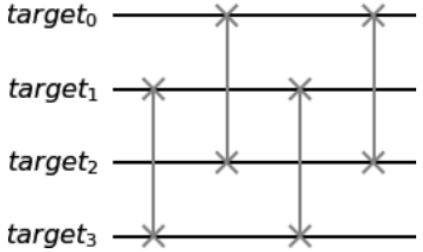
Below is the circuit of controlled U of "2^2 mod 15":



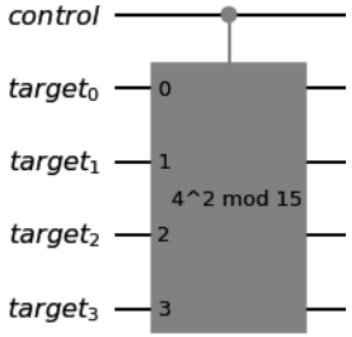
Program 7.9 程式解說[2]:

```
16                      if a in [4, 11]:
17                              U.swap(1,3)
18                              U.swap(0,2)
19                      if a in [7,11,13]:
20                              for j in range(4):
21                                      U.x(j)
22      if show:
23                      print('Below is the circuit of U of '+f'"{a}^{powe
24                      display(U.draw('mpl'))
25      U = U.to_gate()
26      U.name = f'{a}^{power} mod 15'
27      C_U = U.control()
28      return C_U
```

Below is the circuit of U of "4^2 mod 15":



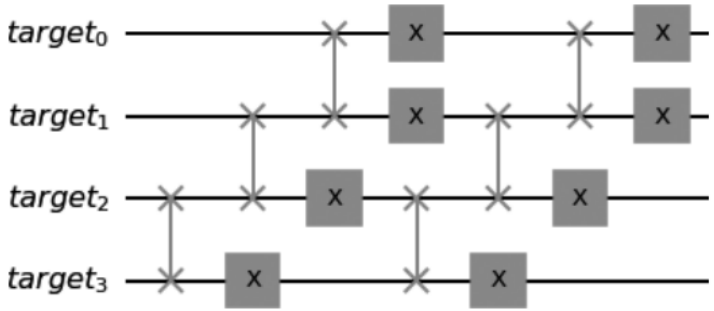
Below is the circuit of controlled U of "4^2 mod 15":



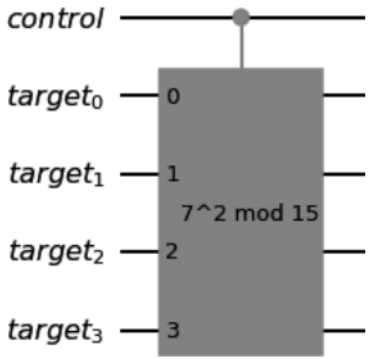
Program 7.9 程式解說[3]:

```
29 power_arg=2
30 for a_arg in [2,4,7,8,11,13]:
31             qrc = QuantumRegister(1,'control')
32             qrt = QuantumRegister(4,'target')
33             qc = QuantumCircuit(qrc,qrt)
34             qc.append(qc_mod15(a_arg, power_arg, show=True),[0,1,2,3,4])
35             print('Below is the circuit of controlled U of '+f'"{a_arg}^{power_arg} mod
       15":')
36             display(qc.draw('mpl'))
```

Below is the circuit of U of "7^2 mod 15":

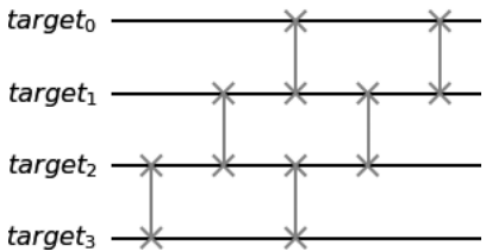


Below is the circuit of controlled U of "7^2 mod 15":

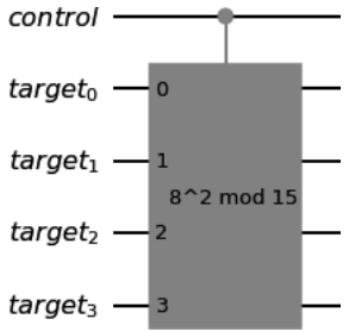


Program 7.9 程式解說[4]:

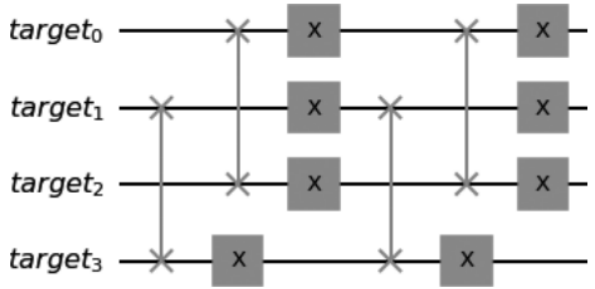
Below is the circuit of U of "8^2 mod 15":



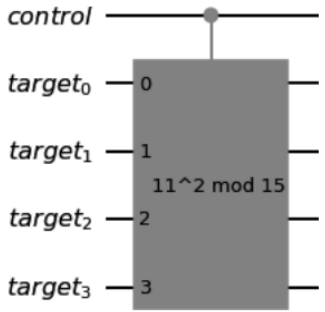
Below is the circuit of controlled U of "8^2 mod 15":



Below is the circuit of U of "11^2 mod 15":

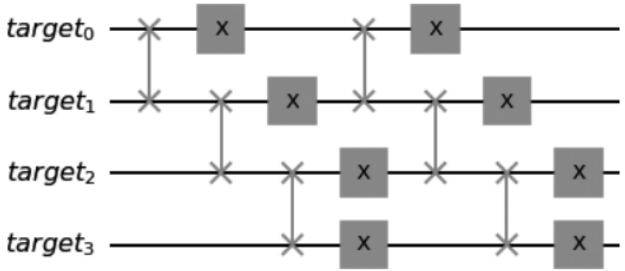


Below is the circuit of controlled U of "11^2 mod 15":

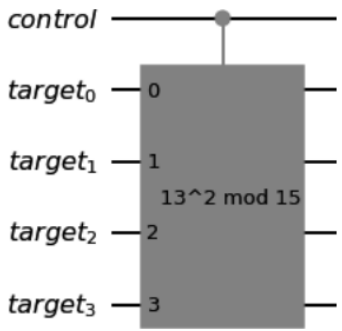


Program 7.9 程式解說[5]:

Below is the circuit of U of "13^2 mod 15":



Below is the circuit of controlled U of "13^2 mod 15":

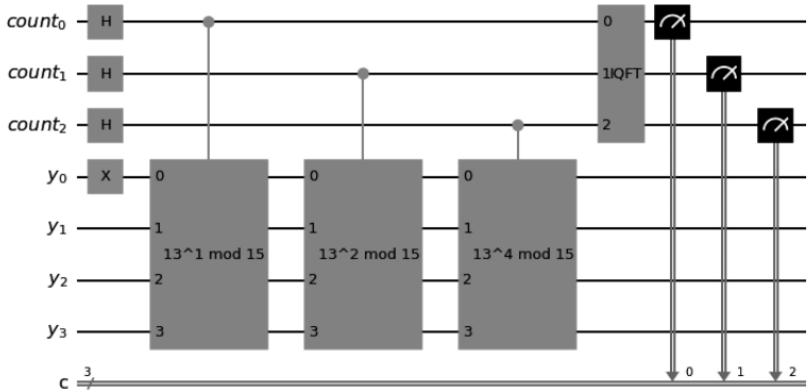


Shor 演算法

- 以上的範例程式定義 qc_mod15 函數，可以回傳么正變換 *U* 的受控 *U* 閘，其中$U|x\rangle = a^x \pmod N$, $N = 15$, $a = 2, 4, 7, 8, 11, 13$。為簡單起見，這個範例程式僅使用連續重複線路的方式但並未使用平方求冪的方式來建構受控 *U* 閘量子線路。
- 以下的範例程式定義 qpf15(quantum period finding for 15) 函數，呼叫剛剛定義的 qc_mod15 函數取得對應么正變換 *U* 的受控量子閘，其中$U|x\rangle = a^x \pmod N$, $N = 15$, $a = 2, 4, 7, 8, 11, 13$。這個受控量子閘可作為建構相位估測線路之用，最後再呼叫函數 iqft 取得逆量子傅立葉變換線路，以便能夠測量量子位元狀態並計算出與 *U* 本徵值對應的相位。

Program 7.10 程式解說:

```
1 #Program 7.10: Define quantum period finding function with N=15
 2 from qiskit import QuantumRegister,ClassicalRegister,QuantumCircuit
 3 def qpf15(count_no,a):
 4             qrc = QuantumRegister(count_no,'count')
 5             qry = QuantumRegister(4,'y') #for input of qc_mod15 gat
 6             clr = ClassicalRegister(count_no,'c')
 7             qc = QuantumCircuit(qrc, qry, clr)
 8             for cbit in range(count_no):
 9                         qc.h(cbit)
10             qc.x(qry[0]) #Set the input of qc_mod15 as |1> with y0 as
11             for cbit in range(count_no): #Add controlled-qc_mod15 g...
12                         qc.append(qc_mod15(a, 2**cbit), [cbit] + list(range(count_no, count_no+4)))
13             qc.append(iqft(count_no).to_gate(label='IQFT'), range(count_no))
14             qc.measure(range(count_no), range(count_no))
15             return qc
16 display(qpf15(count_no=3,a=13).draw('mpl'))
```

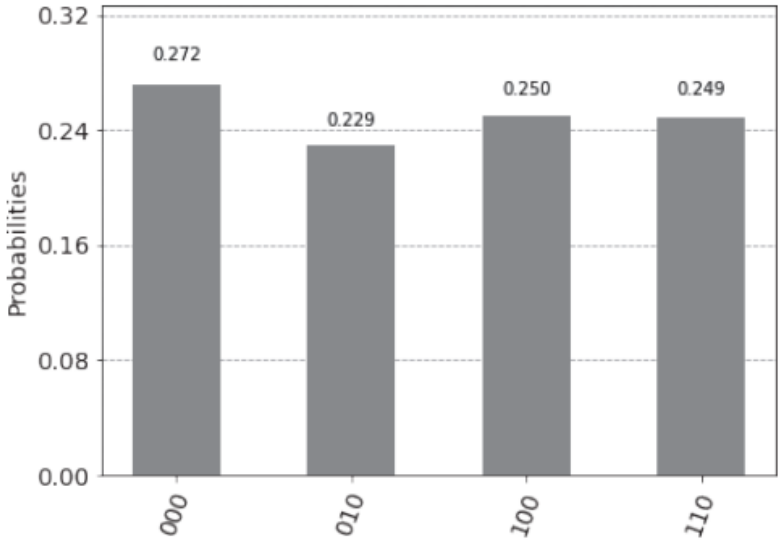


Shor 演算法

- 以下的範例程式呼叫 qpf15 函數，建構對應 $a^x \pmod N$ 么正變換的量子相位估測線路，其中 $N = 15$, $a = 13$，並使用量子電腦模擬器執行量子線路，得到 count 量子位元測量結果，以求出對應的相位：

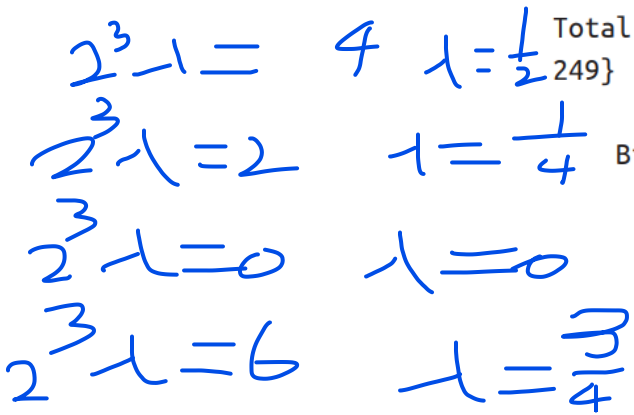
Program 7.11 程式解說[1]:

```
1 #Program 7.11 Run quantum period finding function with N=15
 2 from qiskit import execute
 3 from qiskit.providers.aer import AerSimulator
 4 from qiskit.visualization import plot_histogram
 5 from fractions import Fraction
 6 sim = AerSimulator()
 7 count_no=3
 8 cir = qpf15(count_no=count_no,a=13)
 9 job=execute(cir, backend=sim, shots=1000)
10 result = job.result()
11 counts = result.get_counts(cir)
12 display(plot_histogram(counts))
13 print('Total counts for qubit states are:',counts,'\n')
14 print('%10s %10s %10s %10s %10s' %
('Binary','Decimal','Phase','Fraction','Period'))
```



Program 7.11 程式解說[2]:

```
15 for akey in counts.keys():
16         dec=int(akey,base=2)
17         phase=dec/(2**count_no)
18         frac=Fraction(phase).limit_denominator(15)
19         period=frac.denominator
20         print('%10s %10d %10f %10s %10d' % (akey,dec,phase,frac,period))
```




```
Total counts for qubit states are: {'100': 250, '010': 229, '000': 272, '110':
249}
```

Binary	Decimal	Phase	Fraction	Period
100	4	0.500000	1/2	2
010	2	0.250000	1/4	4
000	0	0.000000	0	1
110	6	0.750000	3/4	4

Shor 演算法

- 上列範例程式使用 3 個量子計數位元估計輸入位元的相位，而量子計數位元的狀態測量結果為 '000'、'010'、'100' 及 '110'，分別為 10 進位的 0、2、4 及 6，其出現機率幾乎相同，大約都是 0.25，因此都列入考慮，希望能藉以找出週期 r。因為 3 個量子計數位元的值介於 0 到 7，總共有 8 個值，因此 4 個測量結果對應的相位的浮點數數值分別為 0/8=0.0、2/8=0.25、4/8=0.5 及 6/8=0.75。
- 事實上，n 個計數位元中記錄的值 c 除以 2^n 之後得到的值，是對應輸入量子位元某一個本徵態的相位 λ。若么正變換的週期為 r，則輸入量子位元一共有 r 個本徵態，它們的相位為 s/r, $0 \le s \le r-1$。也就是說，$c/2^n = \lambda = s/r$, $0 \le s \le r-1$。因此，若能夠將$c/2^n = \lambda$ 透過連分數展開約分為最簡分數 s/r，就可以求出週期 r 可能的值。

Shor 演算法

- 以本範例程式考慮的 $13^y \bmod 15$ 函數為例，經過量子相位估測的結果，測量到的相位為 0/1、1/4、1/2 以及 3/4，因此可以推論這個函數可能的週期為 1、4、2 及4，確實能夠以很高的機率找出正確的週期。

大綱

- 基本概念
- 量子傅立葉變換
- 量子相位估測
- Shor 演算法
- **結語**

結語

- 本章介紹一個解決大整數質因數分解問題的量子演算法—秀爾演算法，這是由美國麻省理工學院（Massachusetts Institute of Technology, MIT）教授彼得·秀爾（Peter Shor）於 1994 年提出的演算法。秀爾演算法將一個正整數 N 進行質因數分解的時間複雜度為 $O((\log N)^2 (\log\log N)(\log\log\log N))$，這是一個多項式時間複雜度。目前最快的大因數分解古典演算法為一般數域篩選（general number field sieve, GNFS），其時間複雜度為 $O(e^{1.9(\log N)^{1/3}(\log\log N)^{2/3}})$ 秀爾演算法相較於一般數域篩選演算法具有指數量級的加速。
- 秀爾演算法先透過古典計算方式將大整數因數分解問題變轉為模冪函數週期尋找問題，並以量子計算方式找出週期，最後再透過古典計算方式針對週期進行檢查以找出大整數的兩個質因數。因為秀爾演算法使用到量子傅立葉變換（及逆量子傅立葉變換）以及量子相位估測，因此本章也特別說明這些技術。

Shor 演算法

- 量子傅立葉變換由 Don Coppersmith 在 1994 年於論文「An approximate Fourier transform useful inquantum factoring」中發表，除了用於秀爾演算法解決大整數因數分解問題之外，也可用於解決隱子群（hidden subgroup）問題，離散對數（discrete logarithm）問題、圖同構（graph isomorphism）問題及最短向量（shortest vector) 等問題。量子相位估測由 Alexei Kitaev 在 1995 於論文「Quantum measurements and the Abelian stabilizer problem」中發表，除了用於秀爾演算法之外，也用於 HHL 演算法，這是 Aram Harrow、Avinatan Hassidim 及 Seth Lloyd 三人在 2009 年於論文「Quantum algorithm for linear systems of equations」中提出的量子演算法，可用於解決線性方程系統（linear systems of equations）問題，與量子機器學習（quantum machine learning）有密切的關聯。

Shor 演算法

- 不論在古典計算方面或是在量子計算方面，秀爾演算法都具有多項式時間複雜度。因此，只要量子電腦的位元數夠多且錯誤率夠低，則秀爾演算法可以在短時間內完成大整數因數分解而破解目前使用最廣的 RSA 密碼系統；因而人們日常生活使用 RSA 加解密機制的系統，如網路金融、電子商務、行動支付、電子合約、數位簽章、公開金鑰基礎建設甚或區塊鏈等系統也將隨之瓦解。很明顯的，秀爾演算法是影響人類生活至深，可以顛覆世界的一個重要演算法。