TP QPE and Shor

Valeran MAYTIE

1 Small Practice

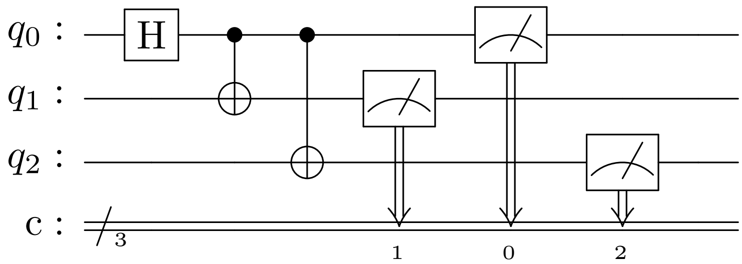

Figure 1: Circuit that calculate $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$

To compute $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$, we created the circuit shown in Figure 1. Firstly we apply an Hadamard Gate on the first qubit to have a 50/50 chance of having it at 1 or 0. If it's equal to one then with a *CNot* gate controlled by the first qubit we inverse q_1 and q_2, so they are all equal to 1. Else nothing change and it remains at 0.

When you run it, you'll find the right measurements the result is roughly 50/50, 000 or 111.

The code for this exercise is shown in Listing 1

```
q = QuantumRegister(3)
c = ClassicalRegister(3)
qc = QuantumCircuit(q,c)

qc.h(q[0])

qc.cnot(q[0],q[1])
qc.cnot(q[0],q[2])

qc.measure(q, c)
```

Listing 1: Code that generates the circuit in Figure 1

2 QPE

We construct the operator $\mathbf{U}$ with this matrix :

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{2i\pi\frac{6}{8}} \end{pmatrix}$$

2.1 Math Questions

1. What is doing this operator ? ('2j' is in Python the complex number $2 \cdot i$)

   The operator $\mathbf{U}$ compute this :

   - $\mathbf{U}|00\rangle = |00\rangle$
   - $\mathbf{U}|01\rangle = |01\rangle$
   - $\mathbf{U}|10\rangle = |10\rangle$
   - $\mathbf{U}|11\rangle = e^{2i\pi\frac{6}{8}}|11\rangle$

2. On how many qubits does it act ?

   This operator act on 2 qubits.

3. What are its eigenvalues/eigenvectors ?

eigenvectors	eigenvalues
$\|00\rangle$	1
$\|01\rangle$	1
$\|10\rangle$	1
$\|11\rangle$	$e^{2i\pi\frac{6}{8}}$

4. For each eigenvector, what should QPE return with 3 bits of precisions, as seen in the course ?

eigenvectors	QPE return
$\|00\rangle$	000
$\|01\rangle$	000
$\|10\rangle$	000
$\|11\rangle$	110

2.2 Implementing QPE

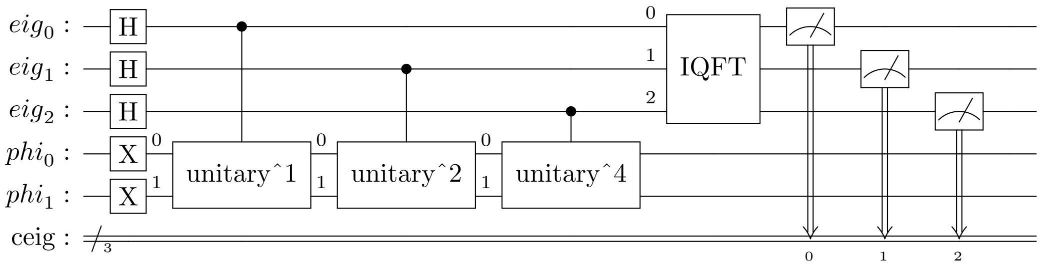

Figure 2: QPE circuit

```
eig = QuantumRegister(size_eig, name="eig")
phi = QuantumRegister(size_phi, name="phi")
ceig = ClassicalRegister(size_eig, name="ceig")
qc = QuantumCircuit(eig,phi,ceig)

qc.x(phi[0])
qc.x(phi[1])
for i in range(0, size_eig):
    qc.h(eig[i])
    qc.append(U.power(2**i).control(), [eig[i], phi[0], phi[1]])

qc.append(QFT(size_eig).inverse(), eig)
qc.measure(eig, ceig)
```

Listing 2: Code that generate the QPE circuit

2.3 Exact result

1. Is it the expected result ?

   Yes, we calculated $110 \equiv \frac{1}{2} + \frac{1}{4} = \frac{6}{8}$ so $\mathbf{U}|11\rangle = e^{2i\pi\theta}|11\rangle$.

   With 3 bits precision θ is equal to $\frac{6}{8}$ seen in Exercise 2.1.1.

2. Change the $\frac{6}{8}$ of the phase of U: use $\frac{1}{8}$, then $\frac{2}{8}$... Is QPE returning the correct answer ?

   Yes, we have 001, 010, ... and 111, when we tested up to $\frac{7}{8}$, because there is enough precision. But then we get the right result modulo 8.

3. Change the precision : use 4 qubits for "eig", and change the fraction in the phase of $\mathbf{U}$ to $\frac{10}{16}$: is QPE indeed returning 10 in binary ?

   We have 10 written in binary. It works because we have enough precision to get the real eigenvalues.

4. Move to 5 bits of precision is it still working ?

   It works, we have $\frac{1}{2} + \frac{1}{8} = \frac{10}{16}$

2.4 Approximate result

The QPE approach calculations with 3-bits precision are given in the table below

value	number of times obtained
000	21
001	34
010	178
011	708
100	39
101	20
110	10
111	14
average / 2^3	0.365

We can see that the average divided by two power of precision is close to the eigenvalue ($\frac{1}{3}$). And the more you increase the accuracy, the closer you get to $\frac{1}{3}$.

2.5 Superposition

By changing the phi initialization to $\frac{1}{\sqrt{2}}(|\phi_1\rangle + |\phi_2\rangle)$ Two calculations are performed in parallel, one to calculate the eigenvalue of eigenvectors $|11\rangle$ and $|00\rangle$.

For the phase it works very well we have :

phi	*eig*	
00	000	498
11	011	526

3 Implementing Shor's algorithm

3.1 Oracle synthesis

$$Mult_{a^p \ mod \ N} : x \mapsto \begin{cases} (a^p \cdot x) \ mod \ N & \text{si } x < N \\ x & \text{si} N \leq x < 2^n \end{cases}$$

```python
def gateMult(a,p,N,n):
  nn = 2 ** n
  M = [[0 for x in range(nn)] for i in range(nn)]
  for x in range(nn):
      if x < N:
          M[((a**p)*x) % N][x] = 1
      else:
          M[x][x] = 1
  U = Operator(M)
  return(UnitaryGate(U))
```

n	time	circuit size (gates)
3	28.2 ms	61
4	120 ms	296
5	527 ms	1341
6	2.46 s	5633
7	11.5 s	23044

Figure 3: Generation of $gateMult(3, 3, 2^n, n)$

1. What are the sizes of the generated circuits ?

   It is written on Figure 3.1.

2. What is the complexity of the circuit size in term of number of qubits ?

   It is exponential.

3. Can you explain why ?

   The matrix as an exponential size in function of the number of qubits ($2^n * 2^n$), so this size has repercussions on the final circuit.

4. What alternate method could you suggest, with what potential drawbacks ?

   You can encode certain types of number, such as powers of two. However, we lose expressiveness.

3.2 Plugging everything together : Shor

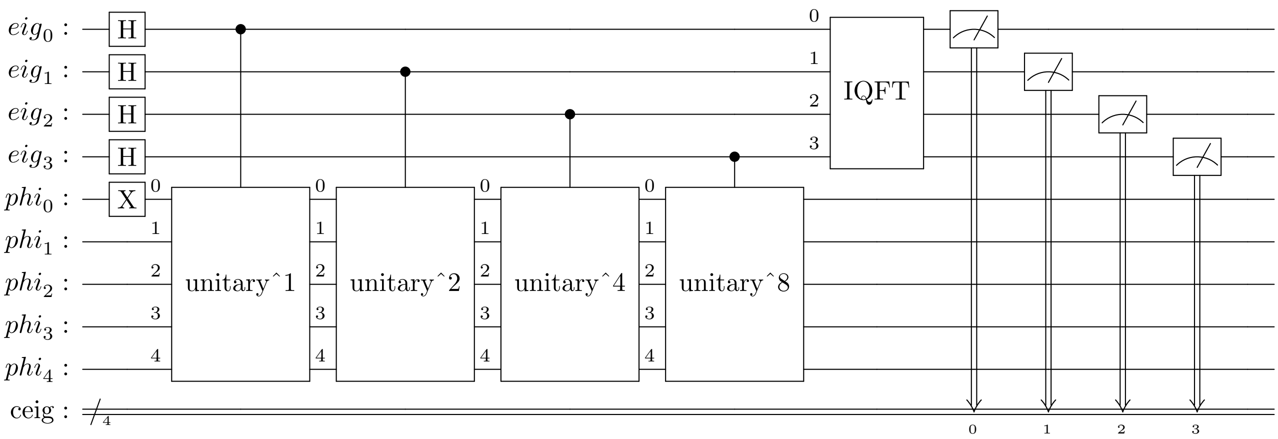

Figure 4: Shor algorithm

1. What is the order r of a mod N (here 7 mod 30) ?

   The order r of 7 mod 30 is 4.

2. On the drawing, where are we supposed to see the values $\frac{s}{r}$? The horizontal axis is graded with integers... To what real numbers between 0 and 1 these correspond to ?

3. Can you infer from the graph the value of r ? Where do you see it on the graph ?

   Yes $r = 4$, it is presented by the period of the keys.

4. Change a and N respectively to 20 and 29. Can you read the value r ? Is it correct ?

   No I can't read the value r

5. The drawing is not very precise... How to make it better ? Try it !

6. Is it still working if you change the value of 'a' and/or 'N' to other values ? Beware not to use too large values for'N'... To get some inspiration, below is the list of possibilities up to 31.

   It works when the result is even.

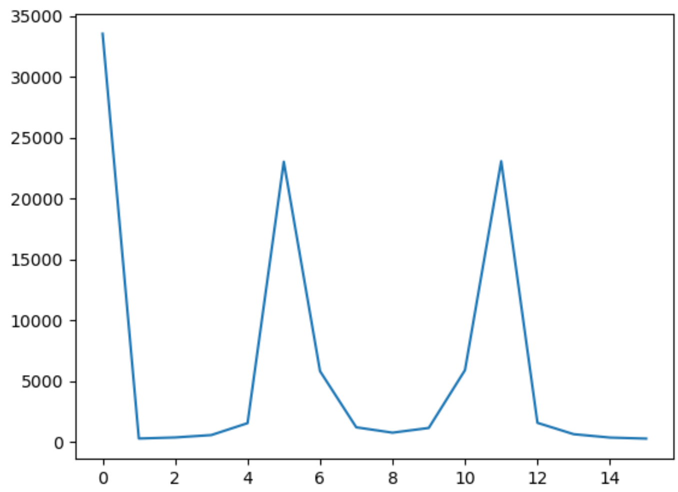

Figure 5: result of shor execution (7 mod 30)

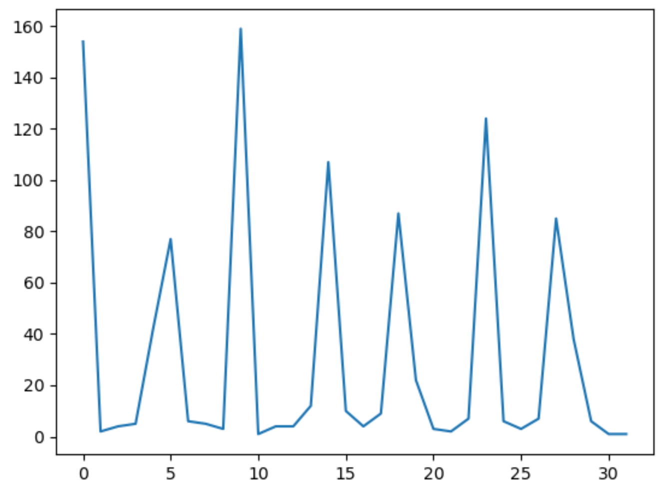

Figure 6: result of shor execution (20 mod 29)