Quantum Phase Estimation

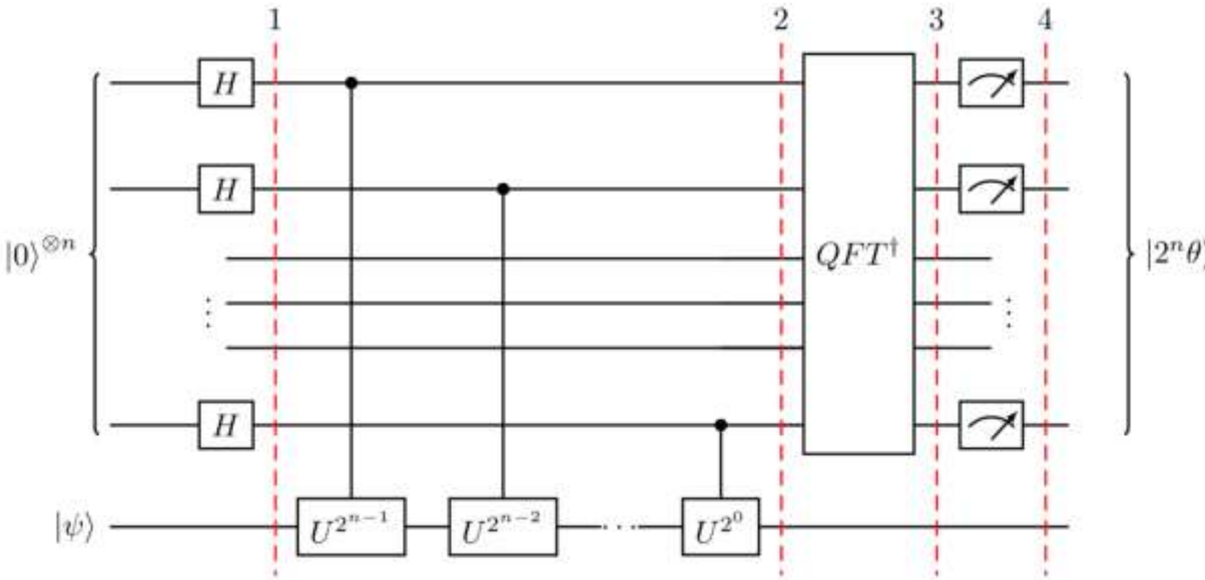
Contents

Quantum phase estimation is one of the most important subroutines in quantum computation. It serves as a central building block for many quantum algorithms. The objective of the algorithm is the following:

Given a unitary operator U, the algorithm estimates θ in $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$. Here $|\psi\rangle$ is an eigenvector and $e^{2\pi i\theta}$ is the corresponding eigenvalue. Since U is unitary, all of its eigenvalues have a norm of 1.

1. Overview

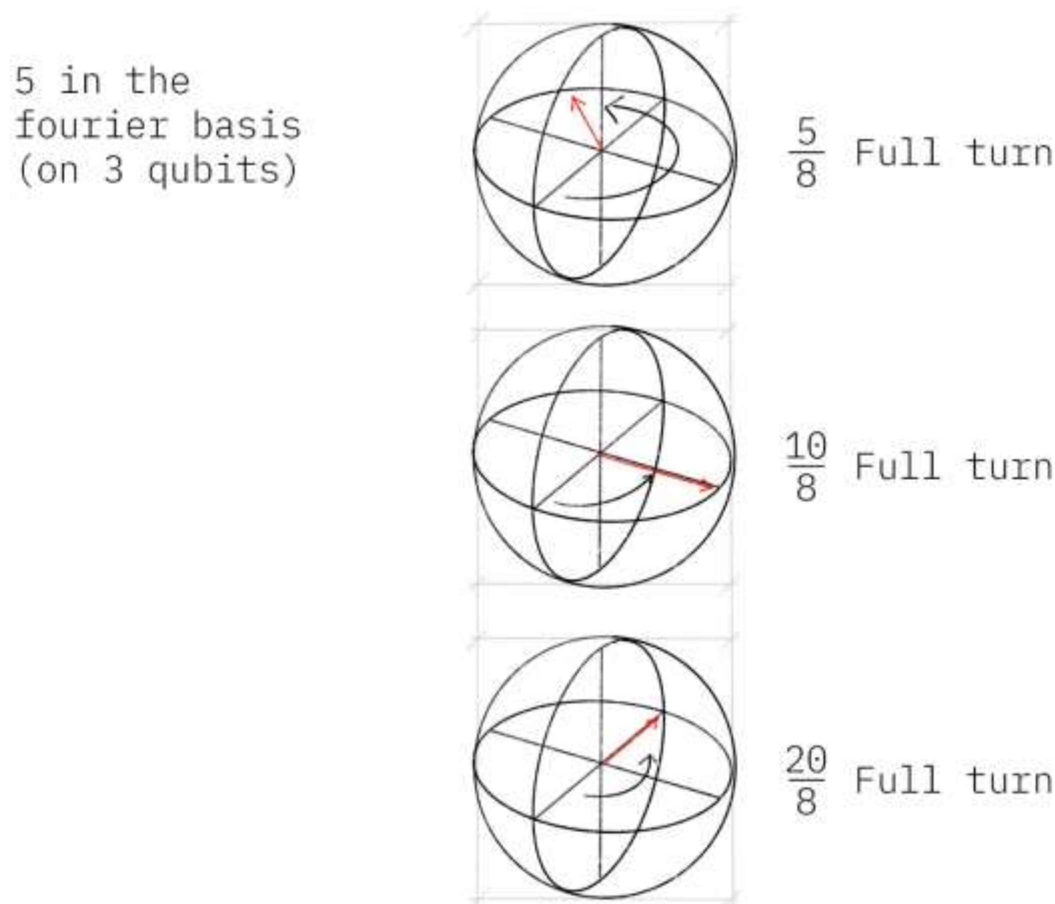
The general quantum circuit for phase estimation is shown below. The top register contains tt 'counting' qubits, and the bottom contains qubits in the state |ψ⟩|ψ⟩:



1.1 Intuition

The quantum phase estimation algorithm uses phase kickback to write the phase of UU (in the Fourier basis) to the tt qubits in the counting register. We then use the inverse QFT to translate this from the Fourier basis into the computational basis, which we can measure.

We remember (from the QFT chapter) that in the Fourier basis the topmost qubit completes one full rotation when counting between 00 and 2t2t. To count to a number, xx between 00 and 2t2t, we rotate this qubit by x2tx2t around the z-axis. For the next qubit we rotate by 2x2t2x2t, then 4x2t4x2t for the third qubit.



When we use a qubit to control the U-gate, the qubit will turn (due to kickback) proportionally to the phase $e^{2i\pi\theta}$. We can use successive CU-gates to repeat this rotation an appropriate number of times until we have encoded the phase theta as a number between 0 and 2^t in the Fourier basis.
Then we simply use $QFT^\dagger$ to convert this into the computational basis.

1.2 Mathematical Foundation

As mentioned above, this circuit estimates the phase of a unitary operator U. It estimates θ in $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$, where $|\psi\rangle$ is an eigenvector and $e^{2\pi i\theta}$ is the corresponding eigenvalue. The circuit operates in the following steps:
i. **Setup**: $|\psi\rangle$ is in one set of qubit registers. An additional set of n qubits form the counting register on which we will store the value $2^n\theta$:
$\psi_0 = |0\rangle^{\otimes n}|\psi\rangle$

ii. **Superposition**: Apply a n-bit Hadamard gate operation $H^{\otimes n}$ on the counting register:

$$\psi_1=\frac{1}{2^{\frac{n}{2}}}(|0\rangle+|1\rangle)^{\otimes n}|\psi\rangle$$

iii. **Controlled Unitary Operations**: We need to introduce the controlled unitary $C-U$ that applies the unitary operator U on the target register only if its corresponding control bit is $|1\rangle$. Since U is a unitary operatory with eigenvector $|\psi\rangle$ such that $U|\psi\rangle=e^{2\pi i\theta}|\psi\rangle$, this means:

$$U^{2^j}|\psi\rangle=U^{2^j-1}U|\psi\rangle=U^{2^j-1}e^{2\pi i\theta}|\psi\rangle=\cdots=e^{2\pi i2^j\theta}|\psi\rangle$$

Applying all the n controlled operations $C-U^{2^j}$ with $0\leq j\leq n-1$, and using the relation $|0\rangle\otimes|\psi\rangle+|1\rangle\otimes e^{2\pi i\theta}|\psi\rangle=(|0\rangle+e^{2\pi i\theta}|1\rangle)\otimes|\psi\rangle$:

$$\psi_2=\frac{1}{2^{\frac{n}{2}}}(|0\rangle+e^{2\pi i\theta 2^{n-1}}|1\rangle)\otimes\cdots\otimes(|0\rangle+e^{2\pi i\theta 2^1}|1\rangle)\otimes(|0\rangle+e^{2\pi i\theta 2^0}|1\rangle)\otimes|\psi\rangle=\frac{1}{2^{\frac{n}{2}}}\sum_{k=0}^{2^n-1}e^{2\pi i\theta k}|k\rangle\otimes|\psi\rangle$$

where k denotes the integer representation of n-bit binary numbers.

iv. **Inverse Fourier Transform**: Notice that the above expression is exactly the result of applying a quantum Fourier transform as we derived in the notebook on Quantum Fourier Transform and its Qiskit Implementation. Recall that QFT maps an n-qubit input state $|x\rangle$ into an output as

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

Replacing x by $2^n\theta$ in the above expression gives exactly the expression derived in step 2 above. Therefore, to recover the state $|2^n\theta\rangle$, apply an inverse Fourier transform on the ancilla register. Doing so, we find

$$|\psi_3\rangle=\frac{1}{2^{\frac{n}{2}}}\sum_{k=0}^{2^n-1}e^{2\pi i\theta k}|k\rangle\otimes|\psi\rangle\xrightarrow{\mathcal{QFT}_n^{-1}}\frac{1}{2^n}\sum_{x=0}^{2^n-1}\sum_{k=0}^{2^n-1}e^{-\frac{2\pi ik}{2^n}(x-2^n\theta)}|x\rangle\otimes|\psi\rangle$$

v. **Measurement**: The above expression peaks near $x=2^n\theta$. For the case when $2^n\theta$ is an integer, measuring in the computational basis gives the phase in the ancilla register with high probability:

$$|\psi_4\rangle=|2^n\theta\rangle\otimes|\psi\rangle$$

For the case when $2^n\theta$ is not an integer, it can be shown that the above expression still peaks near $x=2^n\theta$ with probability better than $4/\pi^2\approx 40\%$ [1].

2. Example: T-gate

Let’s take a gate we know well, the T-gate, and use Quantum Phase Estimation to estimate its phase. You will remember that the T-gate adds a phase of $e^{\frac{i\pi}{4}}$ to the state $|1\rangle$:

$$T|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = e^{\frac{i\pi}{4}}|1\rangle$$

Since QPE will give us θ where:

$$T|1\rangle = e^{2i\pi\theta}|1\rangle$$

We expect to find:

$$\theta = \frac{1}{8}$$

In this example we will use three qubits and obtain an *exact* result (not an estimation!)

2.1 Creating the Circuit

Let's first prepare our environment:

```
#initialization

import matplotlib.pyplot as plt

import numpy as np

import math



# importing Qiskit

from qiskit import IBMQ, Aer

from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister,
execute



# import basic plot tools
```

```
from qiskit.visualization import plot_histogram
```

try

Now, set up the quantum circuit. We will use four qubits -- qubits 0 to 2 as counting qubits, and qubit 3 as the eigenstate of the unitary operator (TT).

We initialize $|\psi\rangle=|1\rangle$|ψ⟩=|1⟩ by applying an XX gate:

```
qpe = QuantumCircuit(4, 3)

qpe.x(3)

qpe.draw()
```

try

Next, we apply Hadamard gates to the counting qubits:

```
for qubit in range(3):

    qpe.h(qubit)

qpe.draw()
```

try

Next we perform the controlled unitary operations. **Remember:** Qiskit orders its qubits the opposite way round to the image above.

```
repetitions = 1

for counting_qubit in range(3):
```

```
    for i in range(repetitions):
        qpe.cu1(math.pi/4, counting_qubit, 3); # This is C-U
    repetitions *= 2
qpe.draw()
```

try

We apply the inverse quantum Fourier transformation to convert the state of the counting register. Here we provide the code for $\text{QFT}^\dagger$:

```
def qft_dagger(circ, n):
    """n-qubit QFTdagger the first n qubits in circ"""
    # Don't forget the Swaps!
    for qubit in range(n//2):
        circ.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            circ.cu1(-math.pi/float(2**(j-m)), m, j)
        circ.h(j)
```

try

We then measure the counting register:

```
qpe.barrier()
```

```python
# Apply inverse QFT

qft_dagger(qpe, 3)

# Measure

qpe.barrier()

for n in range(3):

    qpe.measure(n,n)
```

try

```python
qpe.draw()
```

try

2.2 Results

```python
backend = Aer.get_backend('qasm_simulator')

shots = 2048

results = execute(qpe, backend=backend, shots=shots).result()

answer = results.get_counts()



plot_histogram(answer)
```

try

We see we get one result (`001`) with certainty, which translates to the decimal: `1`. We now need to divide our result (`1`) by 2^n to get θ:

$$\theta = \frac{1}{2^3} = \frac{1}{8}$$

This is exactly the result we expected!

3. Example: Getting More Precision

3.1 The Problem

Instead of a T-gate, let's use a gate with $\theta = \frac{1}{3}$. We set up our circuit as with the last example:

```
# Create and set up circuit

qpe2 = QuantumCircuit(4, 3)



# Apply H-Gates to counting qubits:

for qubit in range(3):

    qpe2.h(qubit)



# Prepare our eigenstate |psi>:

qpe2.x(3)



# Do the controlled-U operations:

angle = 2*math.pi/3

repetitions = 1
```

```
for counting_qubit in range(3):

    for i in range(repetitions):

        qpe2.cu1(angle, counting_qubit, 3);

    repetitions *= 2



# Do the inverse QFT:

qft_dagger(qpe2, 3)



# Measure of course!

for n in range(3):

    qpe2.measure(n,n)



qpe2.draw()
```

try

```
# Let's see the results!

backend = Aer.get_backend('qasm_simulator')

shots = 4096

results = execute(qpe2, backend=backend, shots=shots).result()

answer = results.get_counts()
```

```
plot_histogram(answer)
```

try

We are expecting the result $\theta=0.3333...\theta=0.3333...$, and we see our most likely results are `010(bin) = 2(dec)` and `011(bin) = 3(dec)`. These two results would tell us that $\theta=0.25\theta=0.25$ (off by 25%) and $\theta=0.375\theta=0.375$ (off by 13%) respectively. The true value of $\theta\theta$ lies between the values we can get from our counting bits, and this gives us uncertainty and imprecision.

3.2 The Solution

To get more precision we simply add more counting qubits. We are going to add two more counting qubits:

```
# Create and set up circuit

qpe3 = QuantumCircuit(6, 5)



# Apply H-Gates to counting qubits:

for qubit in range(5):

    qpe3.h(qubit)



# Prepare our eigenstate |psi>:

qpe3.x(5)



# Do the controlled-U operations:

angle = 2*math.pi/3
```

```
repetitions = 1

for counting_qubit in range(5):

    for i in range(repetitions):

        qpe3.cu1(angle, counting_qubit, 5);

    repetitions *= 2



# Do the inverse QFT:

qft_dagger(qpe3, 5)



# Measure of course!

qpe3.barrier()

for n in range(5):

    qpe3.measure(n,n)



qpe3.draw()
```

try

```
### Let's see the results!



backend = Aer.get_backend('qasm_simulator')

shots = 4096
```

```
results = execute(qpe3, backend=backend, shots=shots).result()

answer = results.get_counts()



plot_histogram(answer)
```

try

The two most likely measurements are now `01011` (decimal 11) and `01010` (decimal 10). Measuring these results would tell us θ is:

$$\theta = \frac{11}{2^5} = 0.344,\ \text{or}\ \theta = \frac{10}{2^5} = 0.313$$

These two results differ from $\frac{1}{3}$ by 3% and 6% respectively. A much better precision!

4. Experiment with Real Devices

4.1 Circuit from 2.1

We can run the circuit in section 2.1 on a real device, let's remind ourselves of the circuit:

```
qpe.draw()
```

try

```
# Load our saved IBMQ accounts and get the least busy backend device with
less than or equal to n qubits

IBMQ.load_account()

from qiskit.providers.ibmq import least_busy

from qiskit.tools.monitor import job_monitor

provider = IBMQ.get_provider(hub='ibm-q')
```

```python
backend = provider.get_backend('ibmq_vigo')



# Run with 2048 shots

shots = 2048

job = execute(qpe, backend=backend, shots=2048, optimization_level=3)

job_monitor(job)
```

try

```
Job Status: job has successfully run
```

```python
# get the results from the computation

results = job.result()

answer = results.get_counts(qpe)



plot_histogram(answer)
```

try

We can hopefully see that the most likely result is `001` which is the result we would expect from the simulator. Unlike the simulator, there is a probability of measuring something other than `001`, this is due to noise and gate errors in the quantum computer.

5. Exercises

1. Try the experiments above with different gates (CNOTCNOT, SS, T†T†), what results do you expect? What results do you get?

2. Try the experiment with a YY-gate, do you get the correct result? (Hint: Remember to make sure $|\psi\rangle|\psi\rangle$ is an eigenstate of YY!)

6. Looking Forward

The quantum phase estimation algorithm may seem pointless, since we have to know $\theta\theta$ to perform the controlled-UU operations on our quantum computer. We will see in later chapters that it is possible to create circuits for which we don't know $\theta\theta$, and for which learning theta can tell us something very useful (most famously how to factor a number!)

7. References

[1] Michael A. Nielsen and Isaac L. Chuang. 2011. Quantum Computation and Quantum Information: 10th Anniversary Edition (10th ed.). Cambridge University Press, New York, NY, USA.

8. Contributors

03/20/2020 — Hwajung Kang (@HwajungKang) — Fixed inconsistencies with qubit ordering

```
import qiskit

qiskit.__qiskit_version__
```

try

```
{'qiskit-terra': '0.14.2',

 'qiskit-aer': '0.5.2',

 'qiskit-ignis': '0.3.3',

 'qiskit-ibmq-provider': '0.7.2',

 'qiskit-aqua': '0.7.3',
```

```
 'qiskit': '0.19.6'}
```

They interact and evolve according to
the dipole-dipole potential

$$V_{\mathrm{dip}}(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\left[\frac{\mu_1\cdot\mu_2}{|\mathbf{r}|^3} - 3\frac{(\mu_1\cdot\mathbf{r})(\mu_2\cdot\mathbf{r})}{|\mathbf{r}|^5}\right], \qquad (1)$$

with r the distance between the atoms. We are interested in the limit where the electric field is sufficiently large so that the energy splitting between two adjacent Stark states is much larger than the dipole-dipole interaction. For two atoms in the given initial Stark eigenstate, the diagonal terms of V dip provide an energy shift whereas accumulated phase depends on the precise value of u, i.e. is sensitive to the atomic distance. The probability of loss due to γ is approximately given by p l = 2φγ/u. Furthermore, during the gate operation (i.e. when the the state |rri is occupied) there are large mechanical effects due to the force F. This motivates the following model.

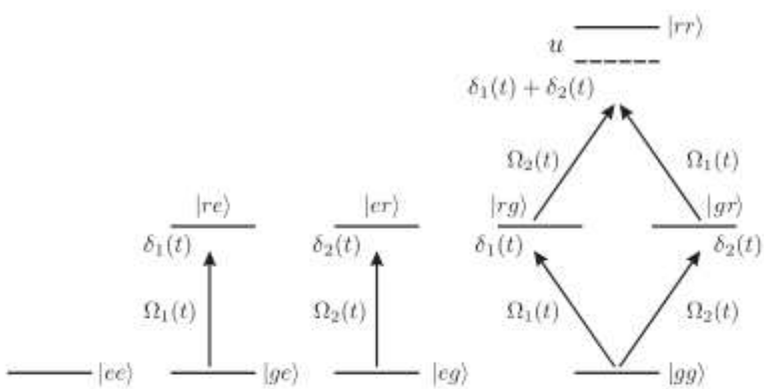


FIG. 2. Schematics of the ideal scheme. The internal state $|g\rangle_j$ is coupled to the excited state $|r\rangle_j$ by the Rabi frequency $\Omega_j(t)$ with the detuning $\delta_j(t)$. The state $|e\rangle_j$ decouples from the evolution of the rest of the system.