Quantum Algorithims & Qiskit

Alexander J. Heilman

August 18, 2021

The Quantum Fourier Transform

The quantum Fourier transform (QFT) is a quantum implementation of the discrete Fourier transform

The Quantum Fourier Transform

The quantum Fourier transform (QFT) is a quantum implementation of the discrete Fourier transform

This means if you give the QFT some state $\sum_n |nk + l\rangle$, the QFT will give back $|k\rangle$.

The Quantum Fourier Transform

The quantum Fourier transform (QFT) is a quantum implementation of the discrete Fourier transform

This means if you give the QFT some state $\sum_n |nk + l\rangle$, the QFT will give back $|k\rangle$.

But what does that look like?

QFT: The gory details

The explict action of the n-qubit QFT on some given basis vector is

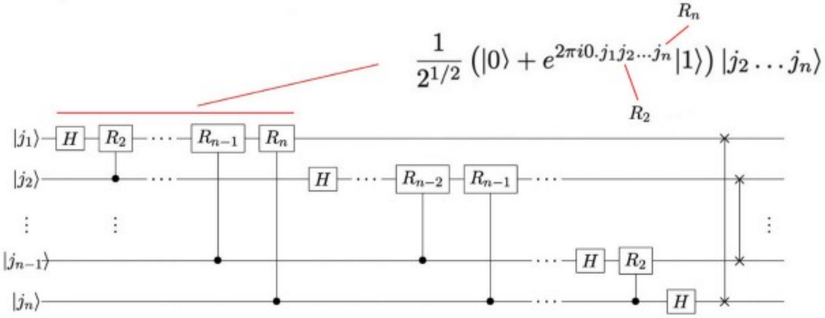
$$|j_1, ..., j_n\rangle \longrightarrow \frac{\left(|0\rangle + e^{2\pi i 0.j_n}|1\rangle\right) \otimes \left(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle\right) \otimes ... \otimes \left(|0\rangle + e^{2\pi i 0.j_1 \dots j_n}|1\rangle\right)}{2^{n/2}}.$$

QFT: The gory details

The explict action of the n-qubit QFT on some given basis vector is

$$|j_1, \ldots, j_n\rangle \longrightarrow$$

$$\frac{\left(|0\rangle + e^{2\pi i 0.j_n}|1\rangle\right) \otimes \left(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle\right) \otimes \ldots \otimes \left(|0\rangle + e^{2\pi i 0.j_1\ldots j_n}|1\rangle\right)}{2^{n/2}}.$$



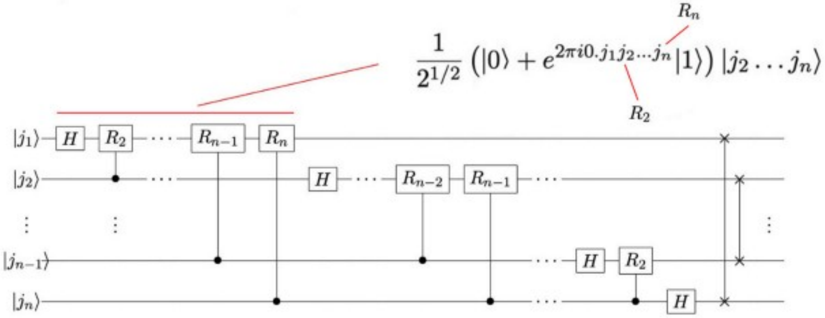
Sorry for asking?

QFT: The gory details

The explict action of the n-qubit QFT on some given basis vector is

$$|j_1, \ldots, j_n\rangle \longrightarrow$$

$$\frac{\left(|0\rangle + e^{2\pi i 0.j_n}|1\rangle\right) \otimes \left(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle\right) \otimes \ldots \otimes \left(|0\rangle + e^{2\pi i 0.j_1 \ldots j_n}|1\rangle\right)}{2^{n/2}}.$$



Sorry for asking? Don't be it's easier to see in matrix form

QFT: The gory details II

For example with $n = 3$ the QFT is

$$\frac{1}{2\sqrt{2}}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega \end{bmatrix}.$$

QFT: The gory details II

For example with $n = 3$ the QFT is

$$\frac{1}{2\sqrt{2}}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega \end{bmatrix}.$$

Not so interesting yet, but it's usefulness will soon be found in phase estimation

Phase Estimation

Phase Estimation takes some given unitary operator and an associated eigenvector and returns the corresponding eigenvalue.

Phase Estimation

Phase Estimation takes some given unitary operator and an associated eigenvector and returns the corresponding eigenvalue.

Phase estimation really returns $|k\rangle$, where the eigenvalue is $\lambda = e^{i\theta}$ with $\theta = k\frac{2\pi}{2^n}$.

Phase Estimation

Phase Estimation takes some given unitary operator and an associated eigenvector and returns the corresponding eigenvalue.

Phase estimation really returns $|k\rangle$, where the eigenvalue is $\lambda = e^{i\theta}$ with $\theta = k\frac{2\pi}{2^n}$.

So how do we do this?

Phase Estimation

Phase Estimation takes some given unitary operator and an associated eigenvector and returns the corresponding eigenvalue.

Phase estimation really returns $|k\rangle$, where the eigenvalue is $\lambda = e^{i\theta}$ with $\theta = k\frac{2\pi}{2^n}$.

So how do we do this? Easy!

Phase Estimation: Not so bad!

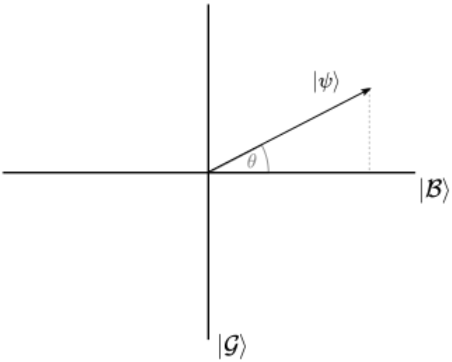
show circuit here with annotated state vector evolution, compare resulting statevector to QFT output

Amplitude Amplification

Amplitude estimation takes some statevector partitioned into a good and a bad subspace

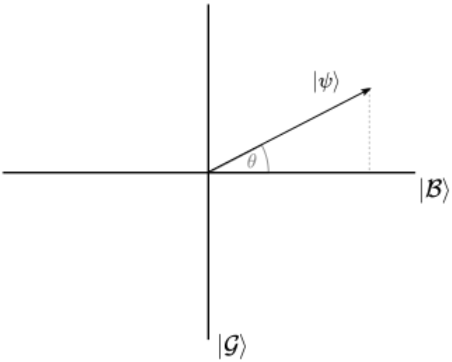
Amplitude Amplification

Amplitude estimation takes some statevector partitioned into a good and a bad subspace



Amplitude Amplification

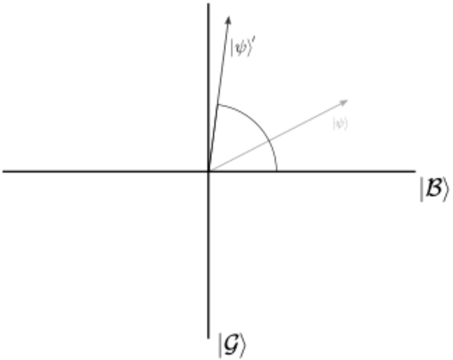
Amplitude estimation takes some statevector partitioned into a good and a bad subspace



$$|\psi\rangle = sin(\theta)|\mathcal{G}\rangle + cos(\theta)|\mathcal{B}\rangle.$$

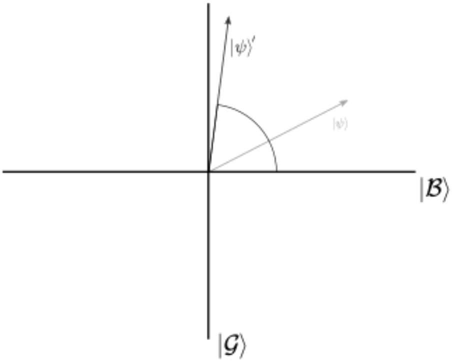
Amplitude Amplification

And returns a statevector nudged towards the good subspace.



Amplitude Amplification

And returns a statevector nudged towards the good subspace.



This is accomplished with consecutive applications of a special operator

Amplitude Amplification: Working principles

Remember $|\psi\rangle = sin(\theta)|\mathcal{G}\rangle + cos(\theta)|\mathcal{B}\rangle$.

Amplitude Amplification: Working principles

Remember $|\psi\rangle = sin(\theta)|\mathcal{G}\rangle + cos(\theta)|\mathcal{B}\rangle$.

We define operators:

Amplitude Amplification: Working principles

Remember $|\psi\rangle = sin(\theta)|\mathcal{G}\rangle + cos(\theta)|\mathcal{B}\rangle$.

We define operators:

1. $\mathcal{S}_{\mathcal{G}} = \mathbb{I} - 2|\mathcal{G}\rangle\langle\mathcal{G}|$

Amplitude Amplification: Working principles

Remember $|\psi\rangle = sin(\theta)|\mathcal{G}\rangle + cos(\theta)|\mathcal{B}\rangle$.

We define operators:

1. $\mathcal{S}_{\mathcal{G}} = \mathbb{I} - 2|\mathcal{G}\rangle\langle\mathcal{G}|$
2. $\mathcal{S}_{\psi} = \mathbb{I} - 2|\psi\rangle\langle\psi|$

Amplitude Amplification: Working principles

Remember $|\psi\rangle = sin(\theta)|\mathcal{G}\rangle + cos(\theta)|\mathcal{B}\rangle$.

We define operators:

1. $\mathcal{S}_{\mathcal{G}} = \mathbb{I} - 2|\mathcal{G}\rangle\langle\mathcal{G}|$
2. $\mathcal{S}_{\psi} = \mathbb{I} - 2|\psi\rangle\langle\psi|$
3. $\mathcal{Q} = -\mathcal{S}_{\psi}\mathcal{S}_{\mathcal{G}}$

Amplitude Amplification: Working principles

Remember $|\psi\rangle = sin(\theta)|\mathcal{G}\rangle + cos(\theta)|\mathcal{B}\rangle$.

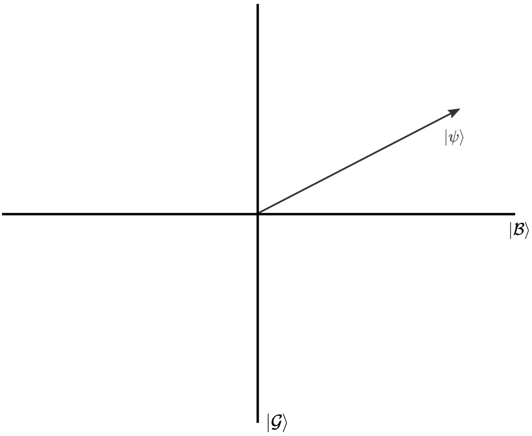
We define operators:

1. $\mathcal{S}_{\mathcal{G}} = \mathbb{I} - 2|\mathcal{G}\rangle\langle\mathcal{G}|$
2. $\mathcal{S}_{\psi} = \mathbb{I} - 2|\psi\rangle\langle\psi|$
3. $\mathcal{Q} = -\mathcal{S}_{\psi}\mathcal{S}_{\mathcal{G}}$

Let's see what they do!

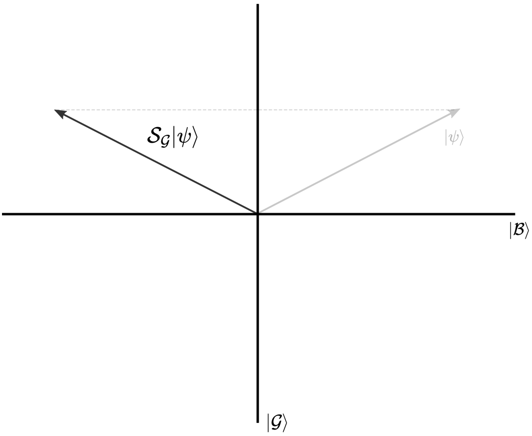
Amplitude Amplification: Visualization I

First apply the operator $\mathcal{S}_{\mathcal{G}}$,



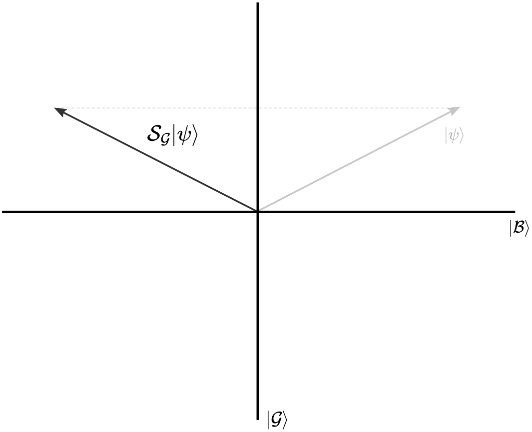
Amplitude Amplification: Visualization I

First apply the operator $\mathcal{S}_{\mathcal{G}}$,



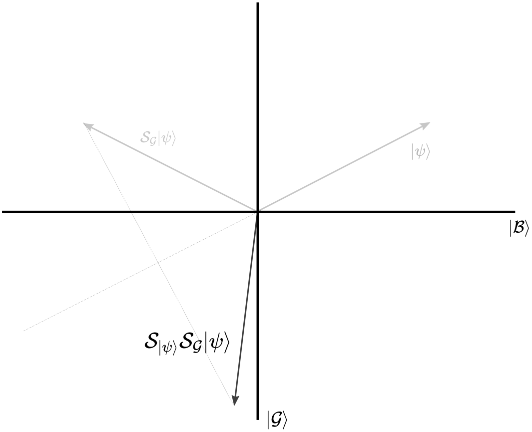
Amplitude Amplification: Visualization II

Then apply the operator $\mathcal{S}_\psi$,



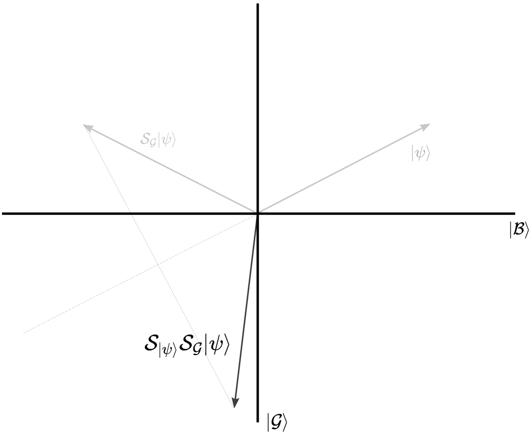
Amplitude Amplification: Visualization II

Then apply the operator $\mathcal{S}_\psi$,



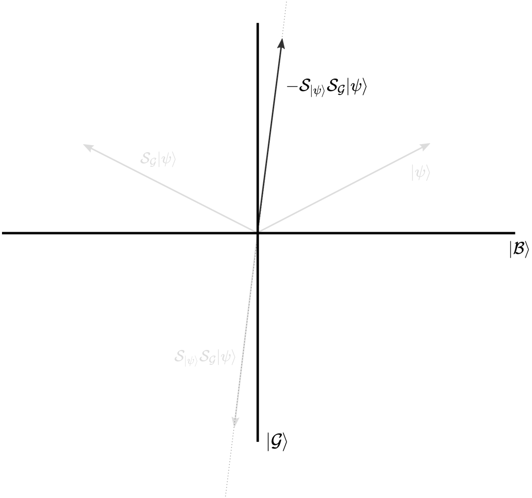
Amplitude Amplification: Visualization III

Now, negate the resulting statevector



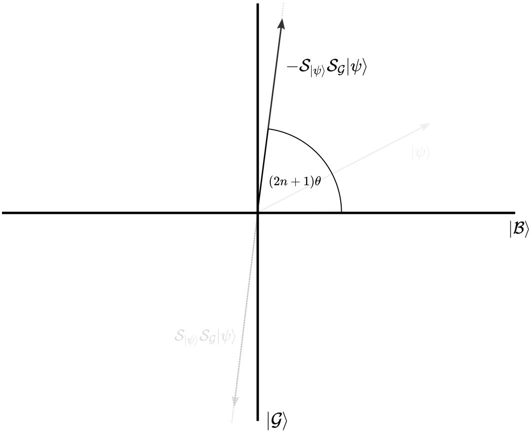
Amplitude Amplification: Visualization III

Now, negate the resulting statevector



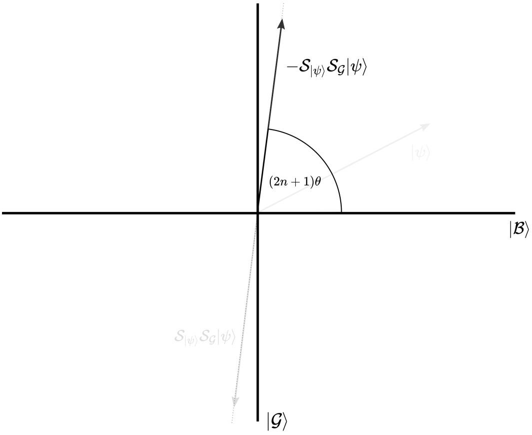
Amplitude Amplification: Visualization IV

And voilá!



Amplitude Amplification: Visualization IV

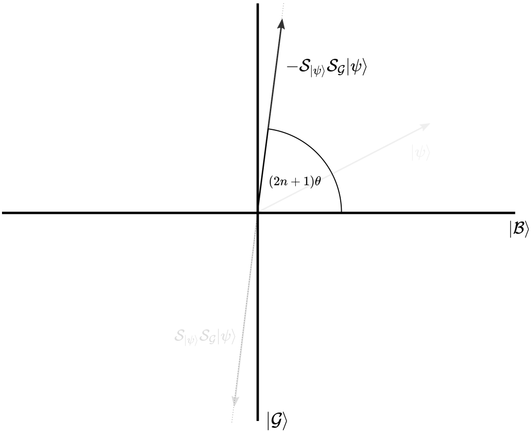
And voilá!



Though, we'd need to know *n* as well

Amplitude Amplification: Visualization IV

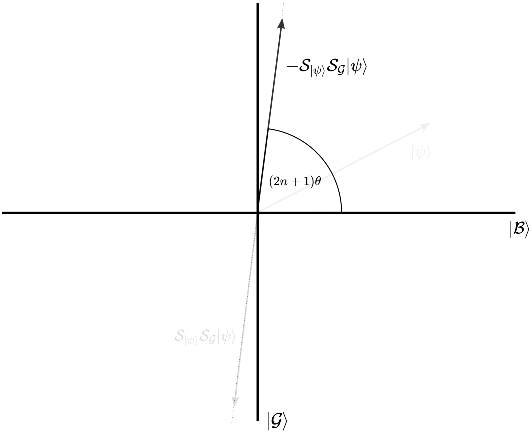
And voilá!



Though, we'd need to know n as well. Ideally $\lfloor \frac{\pi}{4\theta} \rfloor$

Amplitude Amplification: Visualization IV

And voilá!



Though, we'd need to know n as well. Ideally $\lfloor \frac{\pi}{4\theta} \rfloor$

Now, let's do some coding!

Qiskit

Qiskit is a quantum software devlopment kit with a python front-end partially developed by IBM.

Qiskit

Qiskit is a quantum software devlopment kit with a python front-end partially developed by IBM.

Features

1. Simulate + visualize quantum circuits you create yourself

Qiskit

Qiskit is a quantum software devlopment kit with a python front-end partially developed by IBM.

Features

1. Simulate + visualize quantum circuits you create yourself
2. Compatible with IBM's current quantum computers

Qiskit

Qiskit is a quantum software devlopment kit with a python front-end partially developed by IBM.

Features

1. Simulate + visualize quantum circuits you create yourself
2. Compatible with IBM's current quantum computers
3. Vibrant and active online community

Running the QFT

Let's run the QFT on some states.

Running the QFT

Let's run the QFT on some states. We'll use a statevector simulation.

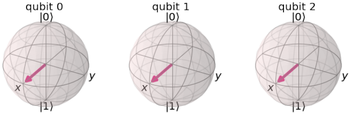
Running the QFT

Let's run the QFT on some states. We'll use a statevector simulation.The local evolution of the qubits is depicted in the bloch spheres below:

Running the QFT

Let's run the QFT on some states. We'll use a statevector simulation.The local evolution of the qubits is depicted in the bloch spheres below:

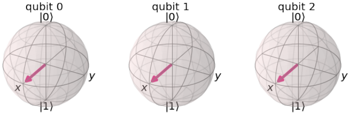
Input: $|000\rangle \rightarrow$



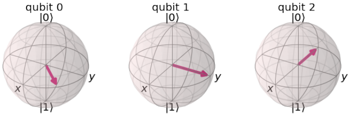
Running the QFT

Let's run the QFT on some states. We'll use a statevector simulation.The local evolution of the qubits is depicted in the bloch spheres below:

Input: $|000\rangle \rightarrow$



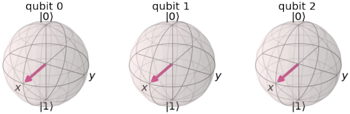
Input: $|001\rangle \rightarrow$



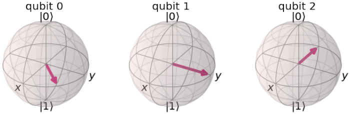
Running the QFT

Let's run the QFT on some states. We'll use a statevector simulation.The local evolution of the qubits is depicted in the bloch spheres below:

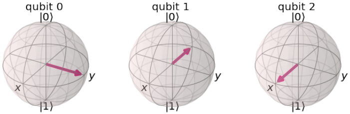
Input: $|000\rangle \rightarrow$



Input: $|001\rangle \rightarrow$



Input: $|010\rangle \rightarrow$



Where's the code?

For the QFT transforms just shown, it's rather simple:

Where's the code?

For the QFT transforms just shown, it's rather simple:

```
from qiskit.circuit.library import QFT
qft = QFT(3)
qft3_000 = execute(qft, backend).result()
plot_bloch_multivector(qft3_000.get_statevector())
```

Where's the code?

For the QFT transforms just shown, it's rather simple:

```
from qiskit.circuit.library import QFT
qft = QFT(3)
qft3_000 = execute(qft, backend).result()
plot_bloch_multivector(qft3_000.get_statevector())
```

Moving forward, I'll just link the code online.

Finding Phases

Let's run the 3-qubit phase estimation algorithim on some gates.

Finding Phases

Let's run the 3-qubit phase estimation algorithim on some gates.

We'll use controlled phase gates

Finding Phases

Let's run the 3-qubit phase estimation algorithim on some gates.

We'll use controlled phase gates, with matrices of the form

$$CP(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\alpha} \end{bmatrix}.$$

Finding Phases

Let's run the 3-qubit phase estimation algorithim on some gates.

We'll use controlled phase gates, with matrices of the form

$$CP(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\alpha} \end{bmatrix}.$$

We already know that $|1\rangle$ is an eigenvector of basic phase gates.

Finding Phases: CP($\pi/4$)

This time, we'll use the qasm simulator

Finding Phases: CP($\pi/4$)

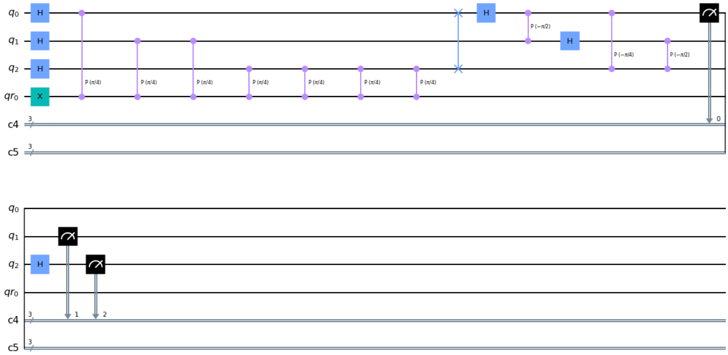
This time, we'll use the qasm simulator. This lets us get counts of simulated measurements at the end

Finding Phases: CP($\pi/4$)

This time, we'll use the qasm simulator. This lets us get counts of simulated measurements at the end. So, let's see the circuit:

Finding Phases: CP($\pi/4$)

This time, we'll use the qasm simulator. This lets us get counts of simulated measurements at the end. So, let's see the circuit:

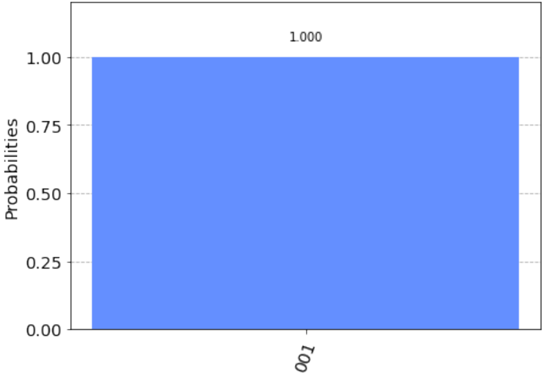


Finding Phases: CP($\pi/4$)

Running the simulation gives a simulated measurement set:

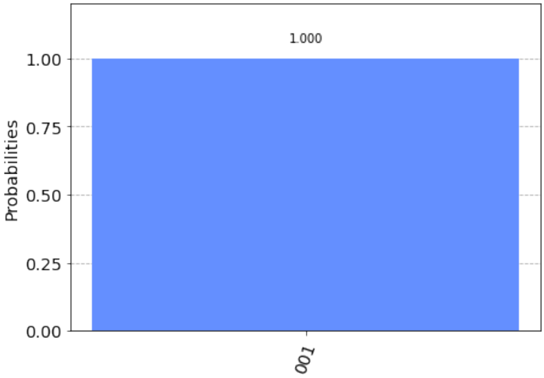
Finding Phases: CP($\pi/4$)

Running the simulation gives a simulated measurement set:



Finding Phases: CP($\pi/4$)

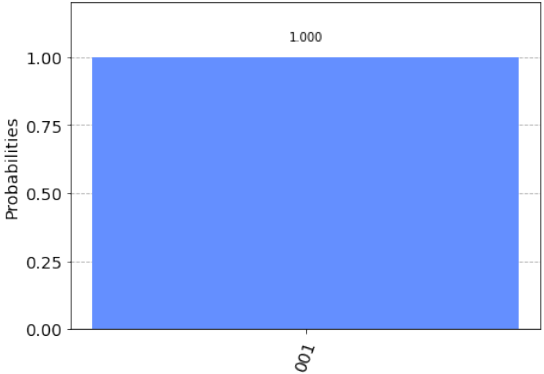
Running the simulation gives a simulated measurement set:



Is this what we would expect?

Finding Phases: CP($\pi/4$)

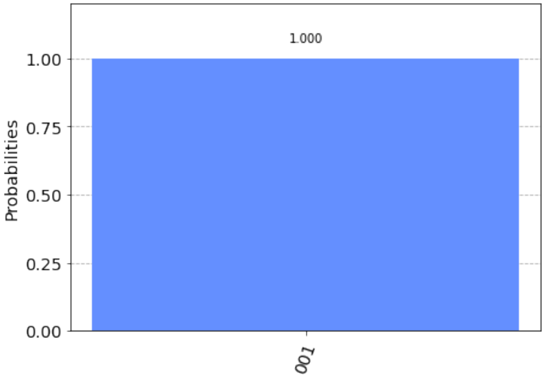
Running the simulation gives a simulated measurement set:



Is this what we would expect? Yes!

Finding Phases: CP($\pi/4$)

Running the simulation gives a simulated measurement set:



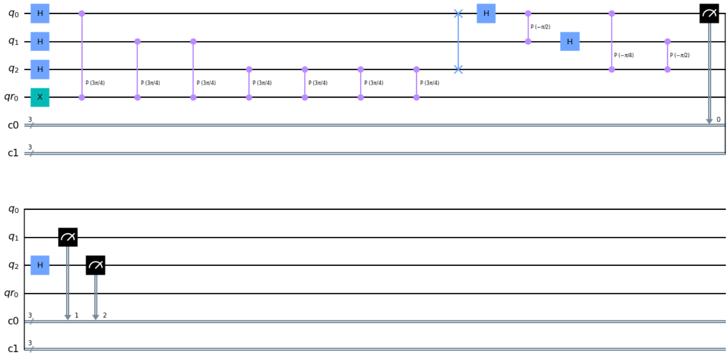
Is this what we would expect? Yes!
As the phase can be encoded perfectly in 3 qubits, we should expect the output vector to be $|k\rangle = |\rangle$

Finding Phases: CP($3\pi/4$)

Running the simulation gives a simulated measurement set:

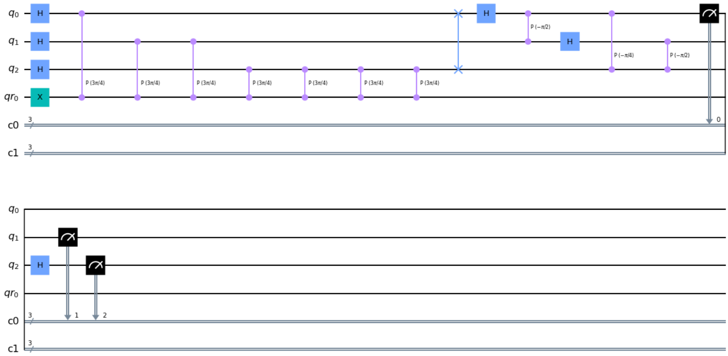
Finding Phases: CP($3\pi/4$)

Running the simulation gives a simulated measurement set:



Finding Phases: CP($3\pi/4$)

Running the simulation gives a simulated measurement set:

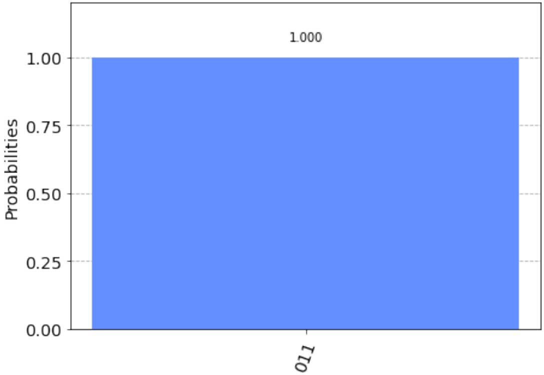


Finding Phases: CP($3\pi/4$)

Running the simulation gives a simulated measurement set:

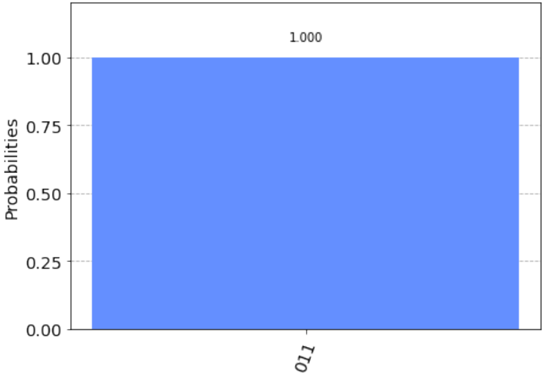
Finding Phases: CP($3\pi/4$)

Running the simulation gives a simulated measurement set:



Finding Phases: CP($3\pi/4$)

Running the simulation gives a simulated measurement set:



References I

Thanks