CPEN 400Q / EECE 571Q Lecture 03
Multi-qubit systems and entanglement

Tuesday 18 January 2022

Announcements

- Assignment 1 available (due 23:59 Thursday 27 Jan)
  - Update forked repo permissions to remove "Students" team
  - Make single PR to master branch on *your* copy of the repo (good idea to do this before adding any of your contents)
  - Error in problem 3 - update to `shots=100000` on devices
- Quiz 1 opens around last 10 mins of class today (work individually; due at 19:30)

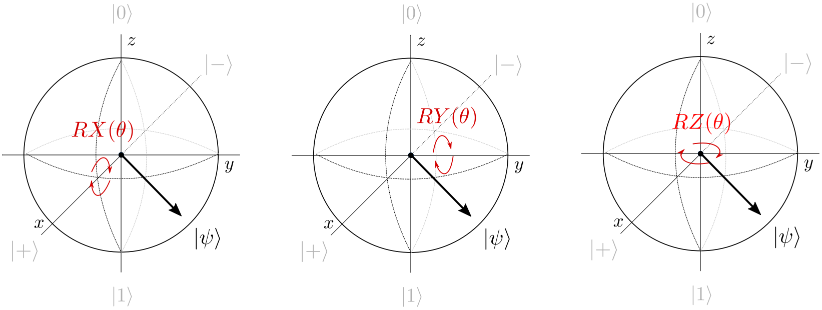
Last time

We learned how to implement quantum circuits in PennyLane, explored a number single-qubit operations, and introduced the notion of *universal gate sets*.

```
import pennylane as qml

dev = qml.device('default.qubit', wires=1, shots=100)

@qml.qnode(dev)
def my_circuit():
    qml.Hadamard(wires=0)
    qml.PauliZ(wires=0)
    qml.PauliX(wires=0)
    return qml.sample()

result = my_circuit()
```

Last time

We saw how qubits can be represented in 3D space on the Bloch sphere, and how unitary operations rotate the Bloch vector.



Fun website: `https://javafxpert.github.io/grok-bloch/`

Image credit: Codebook node I.6

Learning outcomes

- Measure single-qubit expectation values
- Measure a qubit in different bases
- Mathematically describe a system of multiple qubits
- Describe the action of common multi-qubit gates

Measurement: observables and expectation values

Sampling

So far, we've learned how take measurement samples in the computational basis.

```
dev = qml.device('default.qubit', wires=1, shots=100)

@qml.qnode(dev)
def rotate_with_rz(theta):
    qml.Hadamard(wires=0)
    qml.RZ(theta, wires=0)
    return qml.sample()
```

What else can we do?

Measurement outcome probabilities

Compute the measurement outcome probabilities from the results:

```
dev = qml.device('default.qubit', wires=1, shots=100)

@qml.qnode(dev)
def rotate_with_rz(theta):
    qml.Hadamard(wires=0)
    qml.RZ(theta, wires=0)
    return qml.probs()
```

Extract the state

Since we are running on a simulator...

```
# Note that we did NOT specify shots: analytic mode
dev = qml.device('default.qubit', wires=1)

@qml.qnode(dev)
def rotate_with_rz(theta):
    qml.Hadamard(wires=0)
    qml.RZ(theta, wires=0)
    return qml.state()
```

(Can analytically compute probabilities too. But of course we cannot do this with a real device!)

Observables

Generally, we are interested in measuring real, physical quantities. In physics, these are called observables. They are represented by Hermitian matrices. An operator (matrix) H is Hermitian if

$$H = H^\dagger$$

Why Hermitian? The possible measurement outcomes are given by the eigenvalues of the operator, and eigenvalues of Hermitian operators are **real**.

Observables

Example:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Z is Hermitian:

Its eigensystem is

Example:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

X is Hermitian and its (normalized) eigensystem is

Example:

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

Y is Hermitian and its (normalized) eigensystem is

Expectation values

When we measure X, Y, or Z on a state, for each shot we will get one of the eigenstates (/eigenvalues). If we take multiple shots, what do we expect to see *on average*?

Analytically, the **expectation value** of measuring the observable M given the state $|\psi\rangle$ is

$$\langle M \rangle = \langle \psi | M | \psi \rangle.$$

Expectation values: analytical

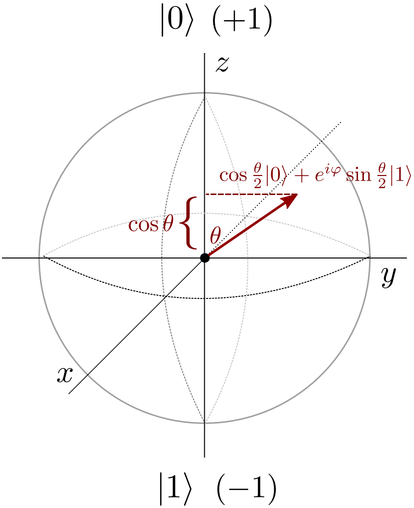
Example: consider the quantum state

$$|\psi\rangle = \frac{1}{2}|0\rangle - i\frac{\sqrt{3}}{2}|1\rangle.$$

Let's compute the expectation value of Y:

Expectation values and the Bloch sphere

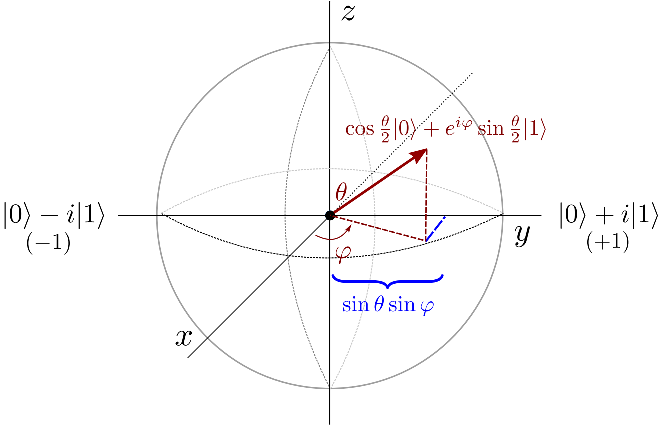
The Bloch sphere offers us some more insight into what a projective measurement is.



$$
\begin{aligned}
|\psi\rangle &= \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle \\
Z|\psi\rangle &= \cos\frac{\theta}{2}|0\rangle - e^{i\varphi}\sin\frac{\theta}{2}|1\rangle \\
\langle\psi|\,Z|\psi\rangle &= \cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2} \\
&= \cos\theta
\end{aligned}
$$

Expectation values and the Bloch sphere

In this picture, we can visualize measurement in different bases by projecting onto different axes.



Exercise: derive this by computing $\langle\psi| Y|\psi\rangle$.

Expectation values: from measurement data

Let's compute the expectation value of Z for the following circuit using 10 samples:

```
dev = qml.device('default.qubit', wires=1, shots=10)

@qml.qnode(dev)
def circuit():
    qml.RX(2*np.pi/3, wires=0)
    return qml.sample()
```

Results might look something like this:

```
[1, 1, 1, 0, 1, 1, 1, 0, 1, 1]
```

Expectation values: from measurement data

The expectation value pertains to the measured eigenvalue; recall Z eigenstates are

$$\lambda_1 = +1, \qquad |\psi_1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\lambda_2 = -1, \qquad |\psi_2\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

So when we observe $|0\rangle$, this is eigenvalue $+1$ (and if $|1\rangle$, -1). Our samples shift from

```
[1, 1, 1, 0, 1, 1, 1, 0, 1, 1]
```

to

```
[-1, -1, -1, 1, -1, -1, -1, 1, -1, -1]
```

Expectation values: from measurement data

The expectation value is the weighted average of this, where the weights are the eigenvalues:

$$\langle Z \rangle = \frac{1 \cdot n_1 + (-1) \cdot n_{-1}}{N}$$

where

- n_1 is the number of $+1$ eigenvalues
- n_{-1} is the number of -1 eigenvalues
- N is the total number of shots

For our example, $\langle Z \rangle = -0.6$.

Expectation values

Let's do this in PennyLane instead:

```
dev = qml.device('default.qubit', wires=1)

@qml.qnode(dev)
def measure_z():
    qml.RX(2*np.pi/3, wires=0)
    return qml.expval(qml.PauliZ(0))
```

So far we've seen 4 ways of extracting information out of a QNode:

1. `qml.state()`
2. `qml.probs(wires=x)`
3. `qml.sample()`
4. `qml.expval(observable)`

The first three all return results of measurements taken with respect to the computational basis; and most hardware only allows for computational basis measurements. How can we measure with respect to *different bases* with that restriction? (and what does that mean?)

What does it mean to measure in a different bases? Projective measurement with respect to a different set of orthonormal states. For example, $\{|+\rangle, |-\rangle\}$ are an orthonormal basis.

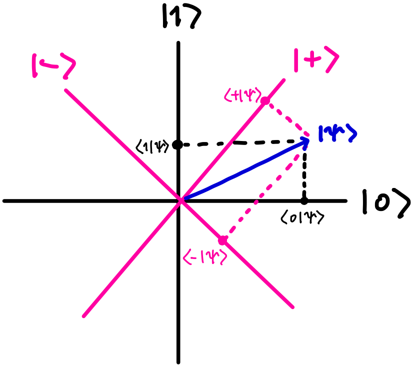


Image credit: Codebook node I.9

Basis rotations

Use a basis rotation to "trick" the quantum computer into measuring in a different basis.

Suppose we want to measure in the Y basis:

$$|i\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + i|1\rangle\right), \quad |-i\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle - i|1\rangle\right).$$

Unitary operations preserve length *and* angles between normalized quantum state vectors.

There exists some unitary transformation that will convert between these eigenvectors, and the eigenvectors of Z (the basis in which we will take the measurement).

Basis rotations

Let's try to turn

$$\begin{aligned} |0\rangle \quad &\rightarrow \quad |i\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + i|1\rangle\right) \\ |1\rangle \quad &\rightarrow \quad |-i\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle - i|1\rangle\right) \end{aligned}$$

At the end of our circuit, we can then apply the reverse (adjoint) of this transformation rotate *back* to the computational basis.

That way, if we measure and observe $|0\rangle$, we know that this was previously $|i\rangle$ in the Y basis (and similarly for $|1\rangle$).

Adjoints

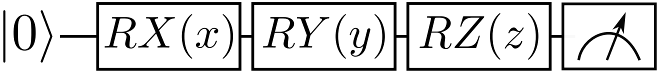
In PennyLane, we can compute adjoints of operations *and* entire quantum functions using `qml.adjoint`:

```
def some_function(x):
    qml.RZ(Z, wires=0)

def apply_adjoint(x):
    qml.adjoint(qml.S)(wires=0)
    qml.adjoint(some_function)(x)
```

`qml.adjoint` is a special type of function called a **transform**. We will cover transforms in more detail around beginning of week 4.

Basis rotations: hands-on

Let's run the following circuit, and measure in the Y basis



Hands-on time...

Mathematics of multi-qubit systems

Multi-qubit systems

Recall this slide from lecture 1...

> **What's so good about qubits?**
>
> - we can make linear combinations of all possible qubit states, i.e., we can put them in **superposition**
> - The size of the mathematical space grows **exponentially** in the number of qubits
> - We can **entangle** multiple qubits, and use these states as a resource in many algorithms

How do we express the mathematical space of multiple qubits?

Hilbert spaces compose under the *tensor product*.

Let

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad B = \begin{pmatrix} e & f \\ g & h \end{pmatrix}.$$

The tensor product of A and B, $A \otimes B$ is

$$A \otimes B = \begin{pmatrix} a \begin{pmatrix} e & f \\ g & h \end{pmatrix} & b \begin{pmatrix} e & f \\ g & h \end{pmatrix} \\ c \begin{pmatrix} e & f \\ g & h \end{pmatrix} & d \begin{pmatrix} e & f \\ g & h \end{pmatrix} \end{pmatrix} = \begin{pmatrix} ae & af & be & bf \\ ag & ah & bg & bh \\ ce & cf & de & df \\ cg & ch & dg & dh \end{pmatrix}$$

Multi-qubit systems

Qubit state vectors are also combined using the *tensor product*:

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 0 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

An n-qubit state is therefore a vector of length 2^n.

Multi-qubit systems

The states $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ are the computational basis vectors for 2 qubits:

$$|00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \quad |01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \quad |10\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \quad |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

We can create arbitrary linear combinations of them as long as the normalization on the coefficients holds.

Same pattern for 3 qubits: $|000\rangle, |001\rangle, \ldots, |111\rangle$.

Multi-qubit systems

The tensor product is linear and distributive, so if we have

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad |\varphi\rangle = \gamma|0\rangle + \delta|1\rangle,$$

then they tensor together to form

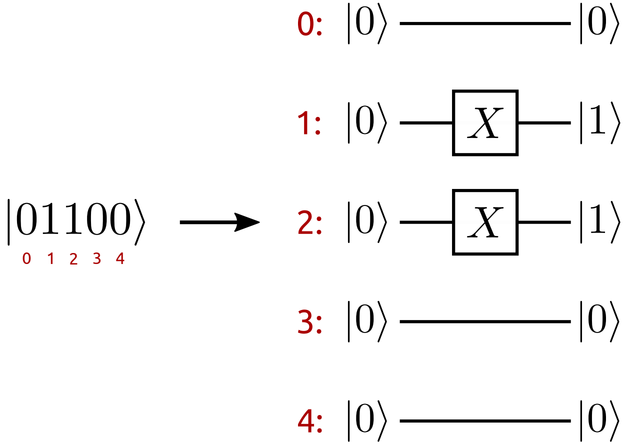
Single-qubit unitary operations also compose under tensor product.

For example, apply U_1 to qubit $|\psi\rangle$ and U_2 to qubit $|\varphi\rangle$:

If an n-qubit ket is a vector with length 2^n, then a unitary acting on n qubits has dimension $2^n \times 2^n$.

Qubit ordering (very important!)

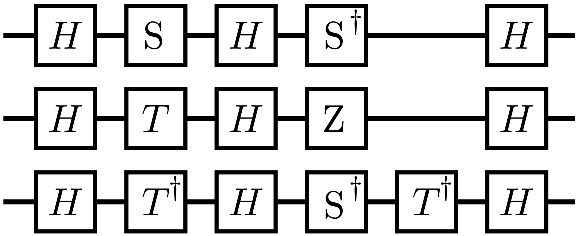
In PennyLane:



(This is different in other frameworks!)

Multi-qubit gates

The few small circuits we've seen so far only involve gates on single qubits:



Surely this isn't all we can do...

Image credit: Xanadu Quantum Codebook I.11

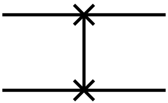
Multi-qubit gates

We can swap the state of two qubits using the SWAP operation.
First define what it does to the basis states...

$$SWAP|00\rangle = |00\rangle$$
$$SWAP|01\rangle = |10\rangle$$
$$SWAP|10\rangle = |01\rangle$$
$$SWAP|11\rangle = |11\rangle$$

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Circuit element:



PennyLane: `qml.SWAP`

More generally,

$$\mathit{SWAP}\left(|\psi\rangle \otimes |\phi\rangle\right) = |\phi\rangle \otimes |\psi\rangle$$

Let's show this. Start by writing

$$\begin{aligned} |\psi\rangle \otimes |\phi\rangle &= \left(\alpha|0\rangle + \beta|1\rangle\right) \otimes \left(\gamma|0\rangle + \delta|1\rangle\right) \\ &= \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle \end{aligned}$$

Now apply the SWAP:

Consider a two-qubit operation U with the following action on the basis states:

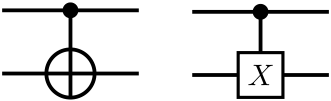
$$
\begin{aligned}
U|00\rangle &= |00\rangle \\
U|01\rangle &= |01\rangle \\
U|10\rangle &= |11\rangle \\
U|11\rangle &= |10\rangle
\end{aligned}
$$

CNOT

CNOT = “controlled-NOT”. A NOT (X) is applied to second qubit only if first qubit is in state $|1\rangle$.

$$
\begin{aligned}
CNOT|00\rangle &= |00\rangle \\
CNOT|01\rangle &= |01\rangle \\
CNOT|10\rangle &= |11\rangle \\
CNOT|11\rangle &= |10\rangle
\end{aligned}
\qquad
CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}
$$

Circuit elements:

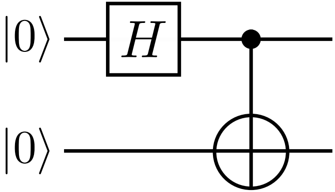


PennyLane: `qml.CNOT`

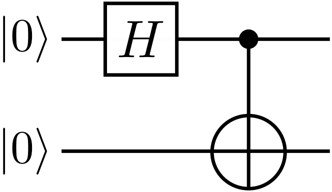
Image credit: Xanadu Quantum Codebook I.12

CNOT hands-on

What does CNOT do with qubits in a superposition?



CNOT hands-on



The output state of this circuit is:

$$CNOT \cdot (H \otimes I)\,|00\rangle = \frac{1}{\sqrt{2}}\,(|00\rangle + |11\rangle)$$

This state is **entangled**!

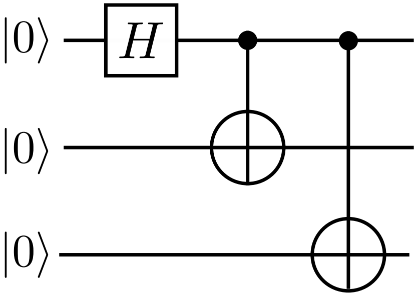
Entanglement

We *cannot express*

$$\frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$$

as a tensor product of two single-qubit states.

Entanglement generalizes to more than two qubits:



Consider the AND of two bits a and b:

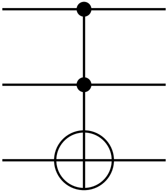
a	b	ab
0	0	0
0	1	0
1	0	0
1	1	1

This gate is *not* reversible: we cannot recover the inputs from the outputs.

But, we can make it reversible by adding one extra bit...

Toffoli

The **Toffoli** implements a reversible AND gate. (It is universal for classical reversible computing).



Controlled-CNOT, or controlled-controlled-NOT.

PennyLane: `qml.Toffoli`

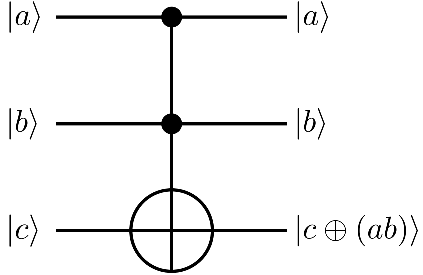
Toffoli

What does it do to the basis states?

$$
\begin{aligned}
TOF|000\rangle &= \\
TOF|001\rangle &= \\
TOF|010\rangle &= \\
TOF|011\rangle &= \\
TOF|100\rangle &= \\
TOF|101\rangle &= \\
TOF|110\rangle &= \\
TOF|111\rangle &=
\end{aligned}
$$

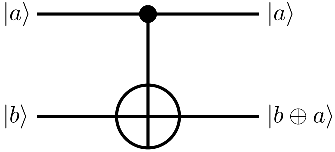
What is actually going on here?

$$
\begin{aligned}
TOF|000\rangle &= |000\rangle \\
TOF|001\rangle &= |001\rangle \\
TOF|010\rangle &= |010\rangle \\
TOF|011\rangle &= |011\rangle \\
TOF|100\rangle &= |100\rangle \\
TOF|101\rangle &= |101\rangle \\
TOF|110\rangle &= |111\rangle \\
TOF|111\rangle &= |110\rangle
\end{aligned}
$$

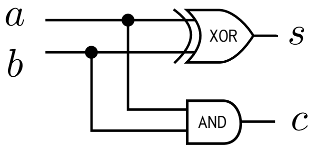


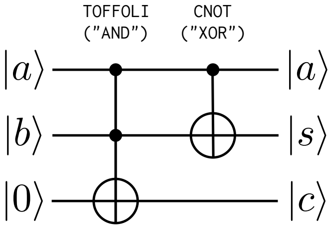
Hands-on: the half-adder

We can interpret CNOT in a similar way.



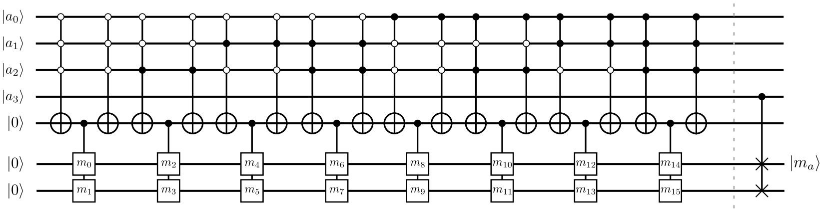
X, CNOT, TOF can be used to create Boolean arithmetic circuits.





Controlled unitary operations

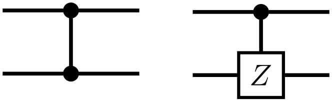
Any unitary operation can be turned into a controlled operation, controlled on any state.



Most common controls are controlled-on-$|1\rangle$ (filled circle), and controlled-on-$|0\rangle$ (empty circle).

Example: controlled-Z (CZ)

What does this operation do?



PennyLane: `qml.CZ`

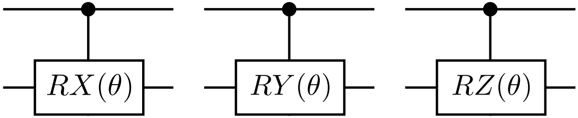
Image credit: Codebook node I.13

Example: controlled rotations (RX, RY, RZ)

Or this one?

$$CRY(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ 0 & 0 & \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

Circuit elements:



PennyLane: `qml.CRX`, `qml.CRY`, `qml.CRZ`

Image credit: Codebook node I.14

There is a pattern here:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad CRY(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ 0 & 0 & \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

More generally,

$$CU = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{00} & U_{01} \\ 0 & 0 & U_{10} & U_{11} \end{pmatrix} = \begin{pmatrix} I_2 & 0_2 \\ 0_2 & U \end{pmatrix}$$

... we don't want to be writing these matrices all the time.

Hands-on: `qml.ctrl`

Remember from earlier, `qml.adjoint`:

```
@qml.qnode(dev)
def my_circuit():
    qml.S(wires=0)
    qml.adjoint(qml.S)(wires=0)
    return qml.sample()
```

There is a similar *transform* that allows us to perform arbitrary controlled operations (or entire quantum functions)!

```
@qml.qnode(dev)
def my_circuit():
    qml.S(wires=0)
    qml.ctrl(qml.S, control=1)(wires=0)
    return qml.sample()
```

Universal gate sets

Last class, we learned that with just

- H and T
- any two of RX, RY, and RZ,

we can implement *any* single-qubit unitary operation up to arbitrary precision.

What about for two qubits?

Universal gate sets

What about for two qubits?

- H, T, and $CNOT$
- any two of RX, RY, RZ, and $CNOT$
- H and TOF

With just 2-3 gates, we can implement *any* two-qubit unitary operation up to arbitrary precision.

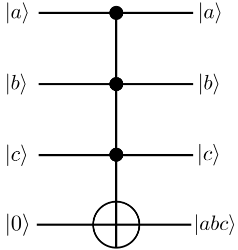
What about three or more qubits? (Same thing!)

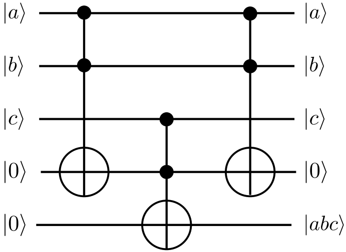
In general, finding such an implementation (*quantum circuit synthesis*, part of the quantum compilation pipeline) is computationally hard.

- sometimes we can do so for small cases (PennyLane has many decompositions pre-programmed)
- sometimes having **auxiliary** qubits around can simplify the decomposition

Auxiliary qubits

Auxiliary qubits are like "scratch", or "work" qubits. They start in state $|0\rangle$, and must be returned to state $|0\rangle$, but can be used to store intermediate results in a computation.





Recap

- Measure single-qubit expectation values
- Measure a qubit in different bases
- Mathematically describe a system of multiple qubits
- Describe the action of common multi-qubit gates

What topics did you find unclear today?

Next time

Content:

- Measuring multi-qubit systems
- Superdense coding
- No-cloning and teleportation

Action items:

1. Continue with Assignment 1 (you can do problem 2 now)

Recommended reading:

- Codebook nodes I.11-I.14
- Nielsen & Chuang 4.3

Quiz time...