CPEN 400Q Lecture 02
Single-qubit systems; introducing PennyLane

Wednesday 8 January 2024

Announcements

- Assignment 1 available later today
- Quiz 1 Monday about lectures 01 and 02 (bring your laptop)
- Tomorrow office hour 2:30-3:00 only
- Next Tuesday tutorial: hands-on activity

Learning outcomes

- Implement single-qubit quantum algorithms in PennyLane
- Describe the behaviour of common single-qubit gates
- Represent the state of a single qubit on the Bloch sphere

Recap from last time

Qubits are physical quantum systems with two **basis states**:

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

Arbitrary states are complex-valued linear combinations

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where $|\alpha|^2 + |\beta|^2 = 1$ and $\alpha, \beta \in \mathbb{C}$.

Qubit states live in **Hilbert space**.

Unitary matrices (gates/operations) modify a qubit's state.

A matrix U is unitary if

$$U U^\dagger = I = U^\dagger U \qquad (U^{-1} = U^\dagger)$$

$(U^*)^T$

They preserve lengths of state vectors and angles between them (to prove on assignment!).

We saw two examples:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

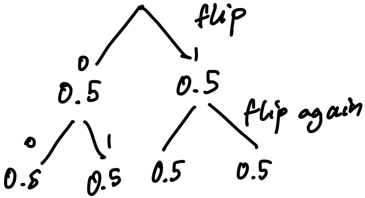
$|0\rangle \to |1\rangle$

$|1\rangle \to |0\rangle$

$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle$

$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle$

What happens if we apply H twice?



$$H|0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \Rightarrow H\left(H|0\rangle\right) = H\left(\frac{1}{\sqrt{2}}|0\rangle\right) + H\left(\frac{1}{\sqrt{2}}|1\rangle\right)$$

$$H\left(H|0\rangle\right) = \frac{1}{\sqrt{2}}\left[\frac{1}{\sqrt{2}}\left(|0\rangle + \cancel{|1\rangle}\right) + \frac{1}{\sqrt{2}}\left(|0\rangle - \cancel{|1\rangle}\right)\right] = |0\rangle$$

$$H\left(H|1\rangle\right) = |1\rangle$$

$$H\begin{pmatrix}0.7\\0.3\end{pmatrix} = H\left(0.7|0\rangle\right) + H\left(0.3|1\rangle\right)$$

Example: Z

The gate

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

does something a little different.

Apply to basis states:

$Z|0\rangle = |0\rangle$ eigenstate :)

$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

$Z|1\rangle = -|1\rangle$

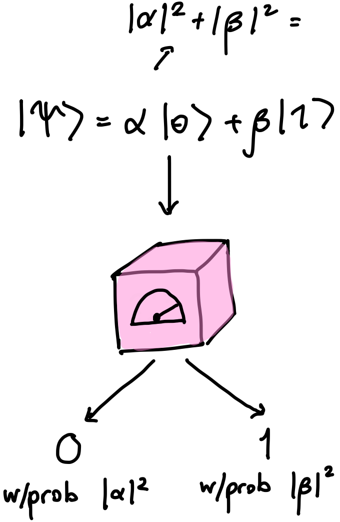
$$Z|+\rangle = \frac{1}{\sqrt{2}}\left(Z|0\rangle + Z|1\rangle\right) = \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right) = |-\rangle$$

Measuring qubits

Applying operations changes the amplitudes in a qubit's state.

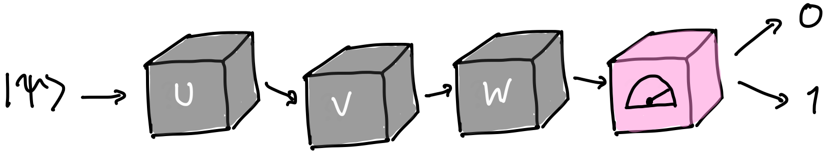
Amplitudes determine probability of observing the qubit in $|0\rangle$ or $|1\rangle$ when measured (it's **probabilistic**!).

Measuring gives us a single bit of information (0 or 1) (generally must run an algorithm many times, i.e., multiple **shots**).



Quantum computing is the act of manipulating the state of qubits in a way that represents solution of a computational problem:

1. **Prepare** qubits in a **superposition**
2. Apply **operations** that **entangle** the qubits and manipulate the amplitudes
3. **Measure** qubits to extract an answer



Exercise: What is the measurement outcome probability of 0 if we apply H, then Z, then X to a qubit starting in $|0\rangle$?

Everything we've done so far is just matrix-vector multiplication...

```
def ket_0():
    return np.array([1.0, 0.0])

def apply_ops(ops, state):
    for op in ops:
        state = np.dot(op, state)
    return state

def measure(state, num_samples=100):
    prob_0 = state[0] * state[0].conj()
    prob_1 = state[1] * state[1].conj()

    samples = np.random.choice(
        [0, 1], size=num_samples, p=[prob_0, prob_1]
    )
    return samples

H = (1/np.sqrt(2)) * np.array([[1, 1], [1, -1]])
X = np.array([[0, 1], [1, 0]])
Z = np.array([[1, 0], [0, -1]])

input_state = ket_0()
output_state = apply_ops([H, X, Z], input_state)
results = measure(output_state, num_samples=10)

print(results)
```

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

Sample NumPy

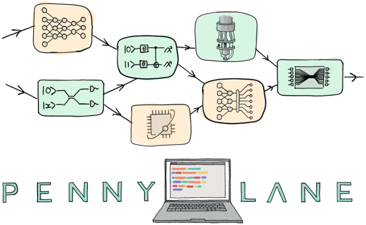
Programming quantum computers

Better to use real quantum software instead. There is a fantastic ecosystem of open-source tools.

```
import pennylane as qml

H = qml.Hamiltonian(...)

dev = qml.device('default.qubit', wires=2)

@qml.qnode(dev)
def quantum_circuit(params):
    qml.RY(params[0], wires=0)
    qml.RY(params[1], wires=1)
    qml.CNOT(wires=[0, 1])
    qml.RY(params[2], wires=0)
    return qml.expval(H)

quantum_circuit([0.1, 0.2, 0.3])
```

PennyLane

PennyLane is a Python framework developed by **Xanadu** (a Toronto-based quantum startup).



GitHub: `https://github.com/PennyLaneAI/PennyLane`
Documentation: `https://pennylane.readthedocs.io/en/stable/`
Demonstrations: `https://pennylane.ai/qml/demonstrations.html`
Discussion Forum: `https://discuss.pennylane.ai/`

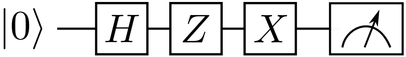
Image credit: `https://pennylane.ai/`

Recall our three quantum gates:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

We can apply these gates to a qubit and express the computation in matrix form, or as a quantum circuit.

$$XZH|0\rangle$$



We can express circuits as **quantum functions** in PennyLane.

Quantum functions

Quantum functions are like normal Python functions, with two special properties:

1. They apply one or more quantum operations

```
import pennylane as qml

def my_quantum_function():
    qml.Hadamard(wires=0) # Apply Hadamard gate to qubit 0
    qml.PauliZ(wires=0)   # Apply Pauli Z gate to qubit 0
    qml.PauliX(wires=0)   # Apply Pauli X gate to qubit 0
    return qml.sample()
```

Q: Why `wires`? A: PennyLane can be used for continuous-variable quantum computing, which does not use qubits.

Quantum functions

Quantum functions are like normal Python functions, with two special properties:

1. They apply one or more quantum operations
2. They return a measurement on one or more qubits

```
import pennylane as qml

def my_quantum_function():
    qml.Hadamard(wires=0)
    qml.PauliZ(wires=0)
    qml.PauliX(wires=0)
    return qml.sample() # Return measurement samples
```

Devices

Quantum functions are executed on **devices**. These can be either *simulators*, or *actual quantum hardware*.

```
import pennylane as qml

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

This creates a device of type `'default.qubit'` with 1 qubit that returns 100 measurement samples when executed.

Quantum nodes

A **QNode (quantum node)** is an object that binds a quantum function to a device, and executes it.

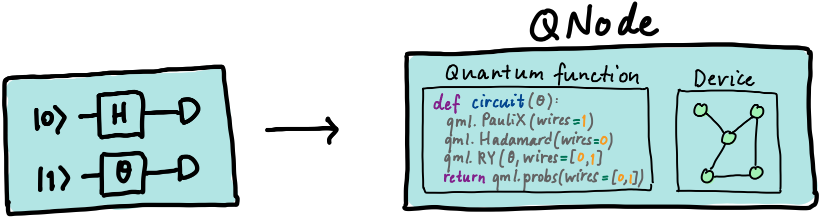


Image credit: `https://pennylane.ai/qml/glossary/quantum_node.html`

Quantum nodes

```python
import pennylane as qml

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

def my_quantum_function():
    qml.Hadamard(wires=0)
    qml.PauliZ(wires=0)
    qml.PauliX(wires=0)
    return qml.sample()
```

With these two components, we can create and execute a QNode.

```python
# Create a QNode
my_qnode = qml.QNode(my_quantum_function, dev)

# Execute the QNode
result = my_qnode()
```

Let's go do it!

You probably have some questions...

1. Where's the state?
   - Inside the device!
2. What happens to the gates?
   - Operations are recorded onto a "tape"
   - The QNode constructs the tape when it is called
   - The tape is then executed on the device.

So far we know the following 3 gates:

$$X, \ H, \ Z$$

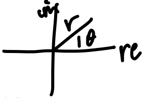
But a general qubit state looks like

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where α and β are *complex numbers* (such that $|\alpha|^2 + |\beta|^2 = 1$).

How do we make the rest?

$re^{i\theta}$



$\cos\phi + i\sin\phi$

Exercise: Consider the states

$$|\psi_1\rangle = \alpha|0\rangle + \beta|1\rangle, \quad |\psi_2\rangle = \alpha e^{i\phi}|0\rangle + \beta e^{i\phi}|1\rangle$$

How does $e^{i\phi}$ affect the measurement outcome probabilities of $|\psi_2\rangle$ compared to $|\psi_1\rangle$?

$$|\psi_1\rangle = \alpha|0\rangle + \beta|1\rangle \Rightarrow |\alpha|^2 \text{ for } 0$$

$$|\psi_2\rangle = \alpha e^{i\phi}|0\rangle + \beta e^{i\phi}|1\rangle \Rightarrow$$

$$|\alpha|^2 = \alpha\alpha^*$$

$$|\alpha e^{i\phi}|^2 = (\alpha e^{i\phi})(\alpha^* e^{-i\phi}) = |\alpha|^2$$

$$|\psi_2\rangle = e^{i\phi}(\alpha|0\rangle + \beta|1\rangle)$$

global phase

Q: How many real numbers are required to fully specify a single-qubit state vector?

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \qquad 3, 4$$

$$= a e^{i\phi}|0\rangle + b e^{i\omega}|1\rangle \qquad a, b, \phi, \omega \text{ real}$$

$$= e^{i\phi}\left(a|0\rangle + b e^{i(\omega-\phi)}|1\rangle\right)$$

$$\sim a|0\rangle + b e^{i\underbrace{(\omega-\phi)}_{\varphi}}|1\rangle \qquad 3 \text{ params}$$

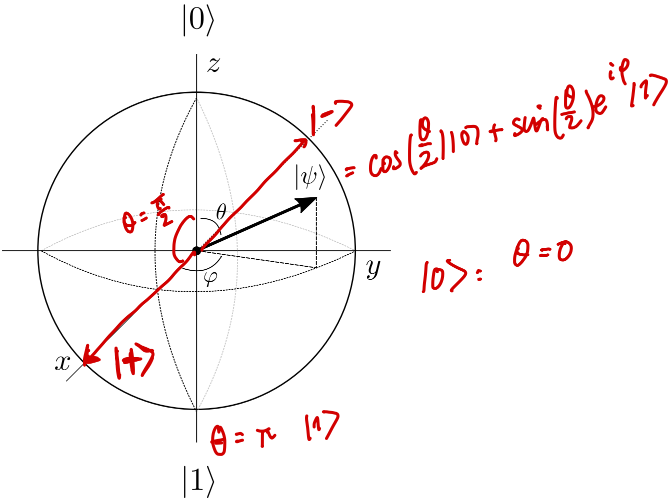
$$= a|0\rangle + b e^{i\varphi}|1\rangle$$

$$|\alpha|^2 + |\beta|^2 = 1$$

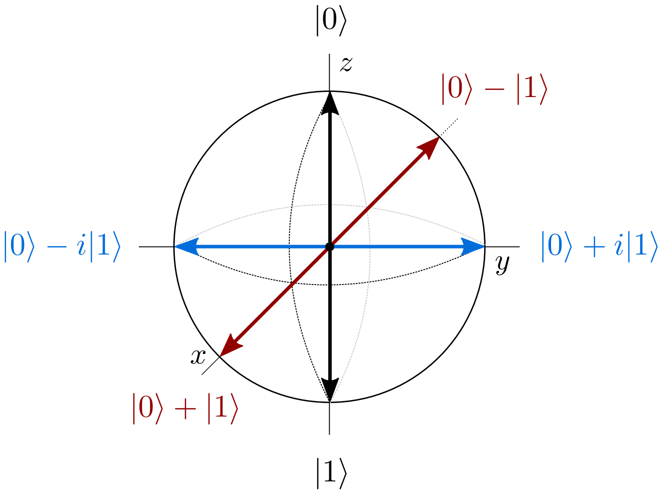
$$a^2 + b^2 = 1 \Rightarrow a = \cos\left(\frac{\theta}{2}\right), \quad b = \sin\left(\frac{\theta}{2}\right)$$

$$\boxed{|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + \sin\left(\frac{\theta}{2}\right)e^{i\varphi}|1\rangle}$$

Introducing the Bloch sphere

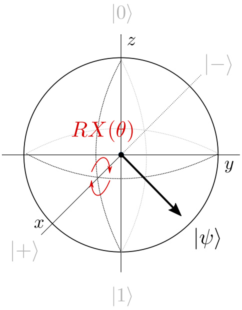


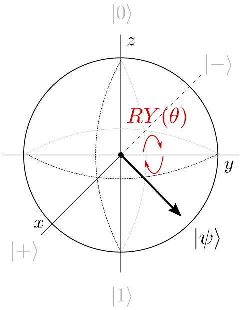
Introducing the Bloch sphere



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

Unitary operations rotate the state vector on the Bloch sphere.





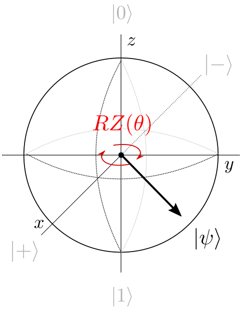
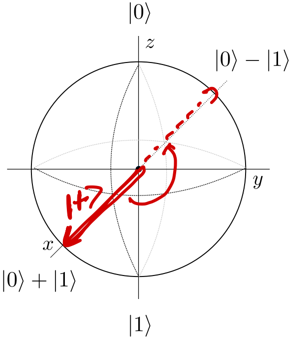


Image credit: Codebook node I.6

Z rotations

$$RZ(\theta) = e^{-i\frac{\theta}{2}Z} = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$$

$$RZ(\pi) : \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix} \sim \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$



In PennyLane, it is called like this:

```
qml.RZ(theta, wires=wire)
```

Exercise: expand out the exponential of Z to obtain the matrix representation.

S and T

Two other special cases: $\theta = \pi/2$, and $\theta = \pi/4$.

$$
\begin{aligned}
S &= RZ(\pi/2) = \begin{pmatrix} e^{-i\frac{\pi}{4}} & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \sim \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \\
T &= RZ(\pi/4) = \begin{pmatrix} e^{-i\frac{\pi}{8}} & 0 \\ 0 & e^{i\frac{\pi}{8}} \end{pmatrix} \sim \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}
\end{aligned}
$$

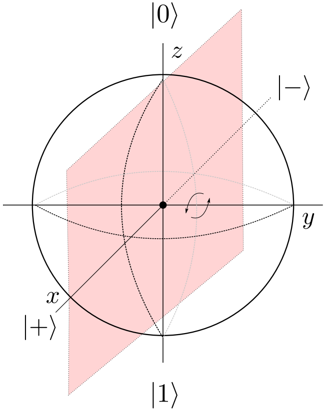
In PennyLane:

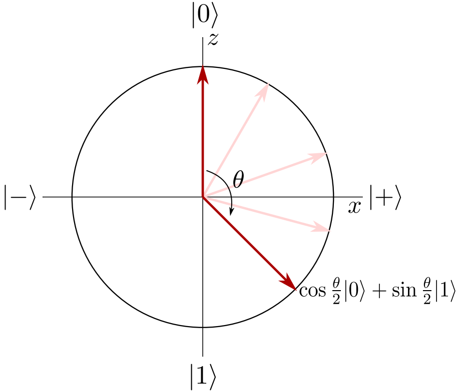
```
qml.PauliZ(wires=wire)
qml.S(wires=wire)
qml.T(wires=wire)
```

S is part of a special group called the **Clifford group**.

T is used in universal gate sets for fault-tolerant QC.

Rotations: RY

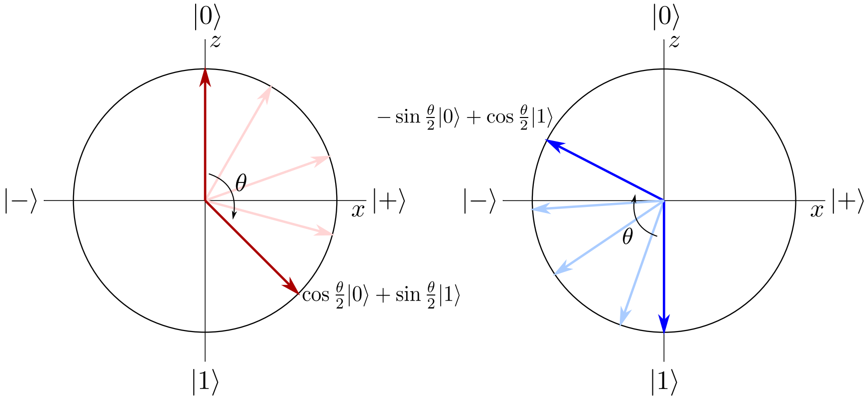




Rotations: RY

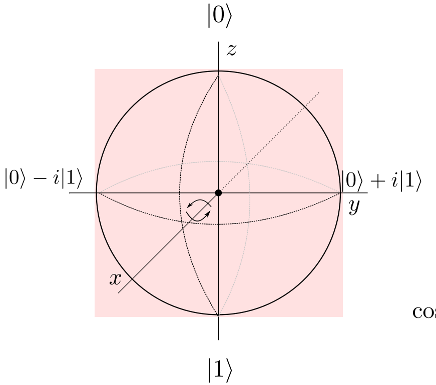
The matrix representation of RY is

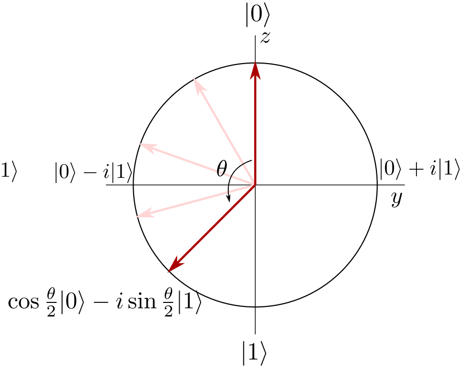
$$RY(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$



RX is similar but has complex components:

$$RX(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$





Pauli rotations

These unitary operations are called **Pauli rotations**.

	Math	Matrix	Code	Special cases (θ)
RZ	$e^{-i\frac{\theta}{2}Z}$	$\begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$	`qml.RZ`	$Z(\pi), S(\pi/2), T(\pi/4)$
RY	$e^{-i\frac{\theta}{2}Y}$	$\begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$	`qml.RY`	$Y(\pi)$
RX	$e^{-i\frac{\theta}{2}X}$	$\begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$	`qml.RX`	$X(\pi), SX(\pi/2)$

Try at home! Use demo code from class as

Exercise: design a quantum circuit that prepares a starting point.

$$|\psi\rangle = \frac{\sqrt{3}}{2}|0\rangle - \frac{1}{2}e^{i\frac{5}{4}}|1\rangle$$

Hint 1: carefully consider RZ and compare with the phase above.

Hint 2: you can also return the state or measurement outcome probabilities in PennyLane:

```
@qml.qnode(dev)
def some_circuit():
    # Gates...
    # return qml.probs(wires=0)
    return qml.state()
```

Try at home!

Exercise: In PennyLane, implement the circuit below

Run your circuit with two different values of θ and take 1000 shots.

How does θ affect the measurement outcome probabilities?

Next time

Content:

- The theory of projective measurements
- Measuring in different bases

Action items:

1. Start looking at Assignment 1
2. Quiz 1 next class

Recommended reading:

- Codebook modules IQC, SQ
- Nielsen & Chuang 4.2, 2.2.3, 2.2.5, 2.2.7