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

Wednesday 10 January 2024

- Assignment 0 due on Monday; Assignment 1 next week
- First quiz on Monday; contents from Monday and today's lectures
- PennyLane v0.34 released yesterday; please update

Last time

We outlined the structure of quantum algorithms:

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

Last time

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

States are written as complex vectors in **Hilbert space**.

Arbitrary states are linear combinations of the basis states:

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

Last time

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

A matrix U is unitary if

$$UU^\dagger = U^\dagger U = \mathbb{1}.$$

They preserve lengths of state vectors and angles between them.

Some examples:

Last time

Measurement at the end of an algorithm is probabilistic.

If we measure a qubit in state

we observe it in

- $|0\rangle$ with probability
- $|1\rangle$ with probability

Let's see some code to simulate a single qubit.

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

PennyLane

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

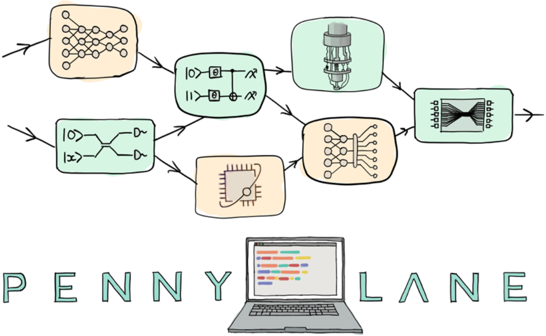


Image credit: `https://pennylane.ai/`

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/`

Its key use case is **differentiable quantum programming** and quantum machine learning; it is also a valuable tool for quantum computing algorithms and applications.

PennyLane

```python
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

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

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

print(my_circuit())
```

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

PennyLane

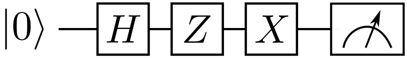
Quantum functions

Recall three of our quantum gates from last time:

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

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

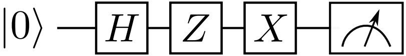
$$XZH|0\rangle$$



Quantum functions

We can also express this circuit as a **quantum function** in PennyLane.

$$XZH|0\rangle$$



```
import pennylane as qml

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

Quantum functions

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

1. 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. Apply one or more quantum operations
2. 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 for anything that is executed.

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

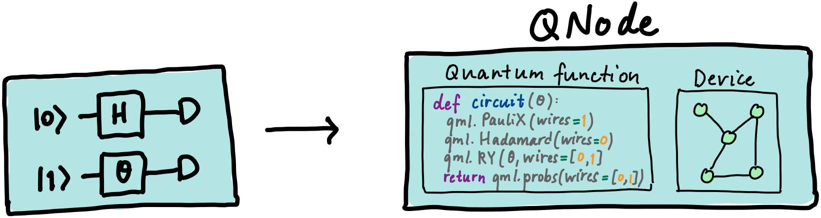


Image credit: `https://pennylane.ai/qml/glossary/quantum_node.html`

Quantum nodes

```
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.

```
# 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 3 gates that do the following:

But a general qubit state looks like

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

How do we make the rest?

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

Global phase

Rewrite $\alpha = ae^{i\phi}$ and $\beta = be^{i\omega}$ with a, b real-valued:

Factor out the $e^{i\phi}$ (a **global phase**):

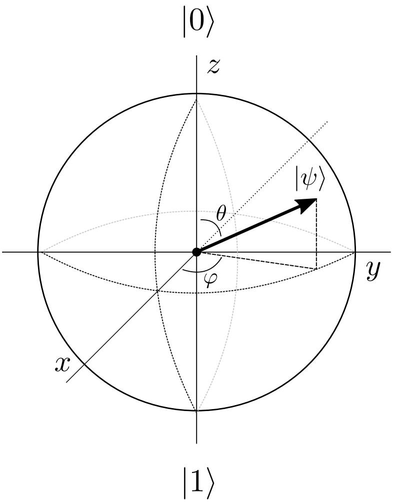
Parametrization of qubit states

Normalization tells us $a^2 + b^2 = 1$. What else looks like this?

We can rewrite as:

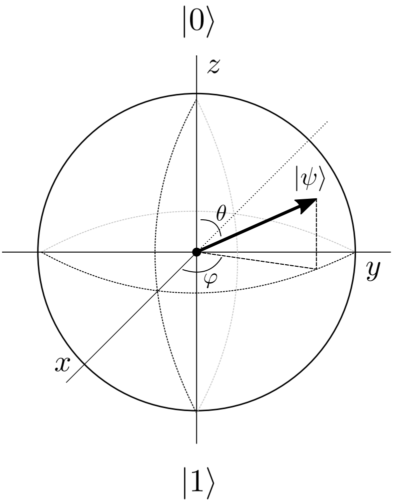
Any single-qubit state can be specified by two angles...

Introducing the Bloch sphere



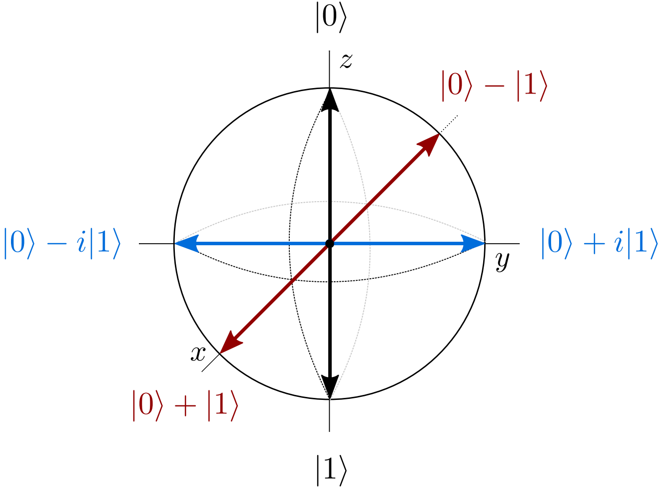
`https://javafxpert.github.io/grok-bloch/`

Introducing the Bloch sphere



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

Introducing the Bloch sphere



Rotations: the Bloch sphere

Unitary operations correspond visually to rotations.

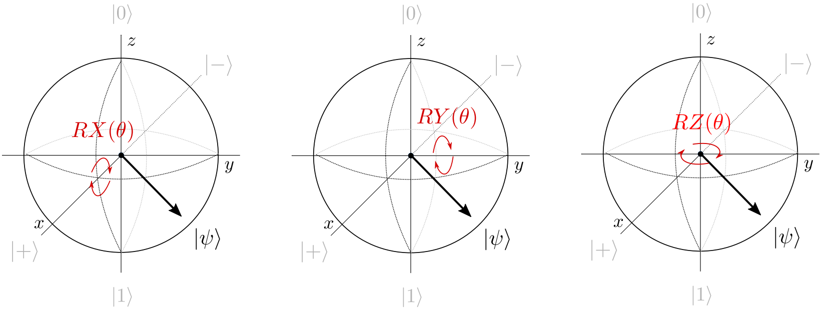
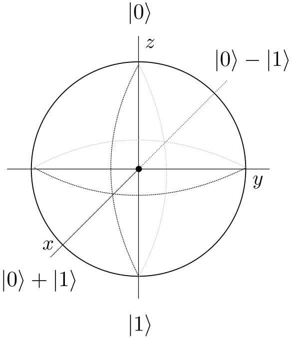


Image credit: Codebook node I.6

Z rotations

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



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 $\theta = \pi/4$.

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

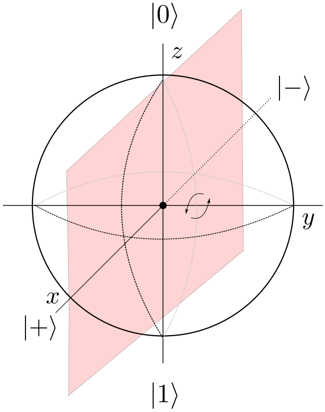
In PennyLane:

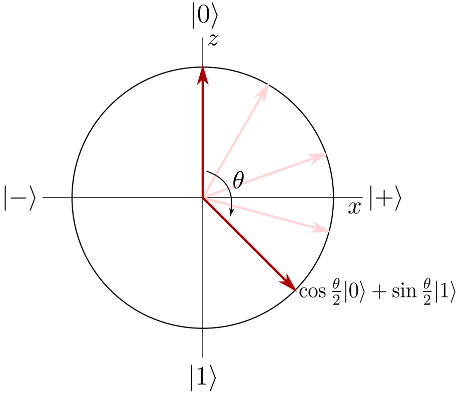
```
qml.S(wires=wire)
qml.T(wires=wire)
```

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

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

Rotations: *RY*

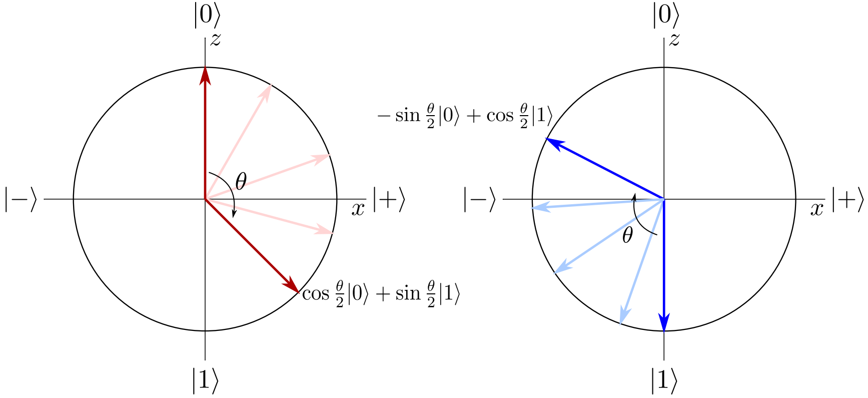




Rotations: *RY*

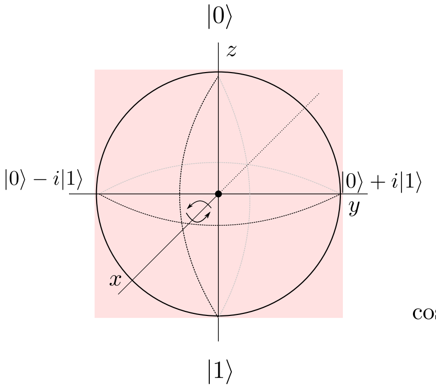
The matrix representation of *RY* is

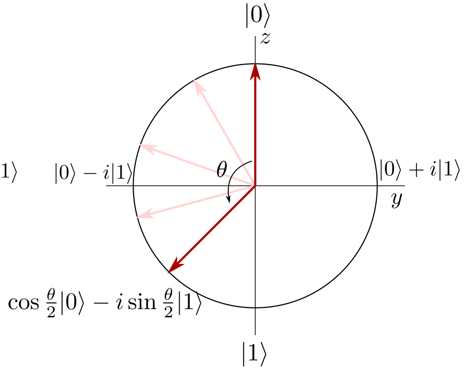
$$RY(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$



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
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`	$X(\pi), SX(\pi/2)$

Exercise: design a quantum circuit to prepare the state

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

Hint: 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()
```

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?

Recap

- 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

Next time

Content:

- The theory of projective measurements
- Measuring in different bases

Action items:

1. Finish Assignment 0 (due Monday evening)
2. Quiz next class

Recommended reading:

- Codebook nodes I.1-I.10
- Nielsen & Chuang 4.2