Codercise I.6.1 — Applying RX

Open related theory

In addition to the RZ rotation, we have also RX, and RY rotations. These parametric gates are available in PennyLane as `qml.RX` and `qml.RY`.

Write a QNode that applies `qml.RX` with an angle of π to one of the computational basis states. What operation is this?

Solution :

```
dev = qml.device("default.qubit", wires=1)
@qml.qnode(dev)
def apply_rx_pi(state):
    """Apply an RX gate with an angle of \pi to a particular basis state.
    Args:
        state (int): Either 0 or 1. If 1, initialize the qubit to state |1>
            before applying other operations.
    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    if state == 1:
        qml.PauliX(wires=0)
    ##################
    qml.RX(np.pi,wires=0)
    ##################
    # APPLY RX(pi) AND RETURN THE STATE
    return qml.state()

print(apply_rx_pi(0))
print(apply_rx_pi(1))
```

```
def apply_rx_pi(state):
    """Apply an RX gate with an angle of \pi to a particular basis state.

    Args:
        state (int): Either 0 or 1. If 1, initialize the qubit to state |1>
            before applying other operations.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    if state == 1:
        qml.PauliX(wires=0)

    ##################
    qml.RX(np.pi,wires=0)
    ##################

    # APPLY RX(pi) AND RETURN THE STATE

    return qml.state()


print(apply_rx_pi(0))
print(apply_rx_pi(1))
```

Reset Code Submit

Correct!

User output

```
[6.123234e-17+0.j 0.000000e+00-1.j]
[0.000000e+00-1.j 6.123234e-17+0.j]
```

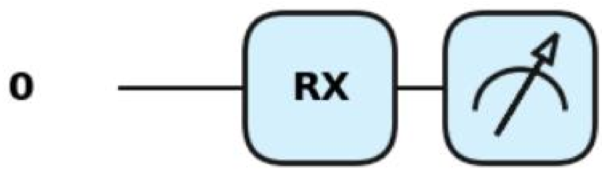
Qiskit Program:

```
import numpy as np
import random
from qiskit.quantum_info import Statevector
import pennylane as qml
import matplotlib.pyplot as plt

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

@qml.qnode(dev)
@qml.qnode(dev)
def apply_rx_pi(state):
    """Apply an RX gate with an angle of \pi to a particular basis state.

    Args:
        state (int): Either 0 or 1. If 1, initialize the qubit to state |1>
            before applying other operations.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    if state == 1:
        qml.PauliX(wires=0)
    ##################
    # APPLY RX(pi) AND RETURN THE STATE
    qml.RX(np.pi,wires=0)
    ##################

    return qml.state()

print(apply_rx_pi(0))
print(apply_rx_pi(1))
```

O/P:

```
[6.123234e-17+0.j 0.000000e+00-1.j]
[0.000000e+00-1.j 6.123234e-17+0.j]
```

```
circuit = qml.QNode(apply_rx_pi, dev)
qml.drawer.use_style("pennylane")
result = qml.draw_mpl(circuit)(0)
plt.show()
```



Codercise I.6.2 — Plotting RX

Open related theory

In the previous exercise, you should have noticed that for this special case, $RX(\pi) = X$ up to a global phase of $-i$. But what does RX do more generally?

The matrix representation of RX is

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

How does this affect the amplitudes when we apply it to a quantum state? Implement a QNode that applies the `qml.RX` operation with parameter θ to a specified basis state. Then, run the code to plot the amplitudes of the $|0\rangle$ and $|1\rangle$ after applying $RX(\theta)$ to the $|0\rangle$ state.

Solution:

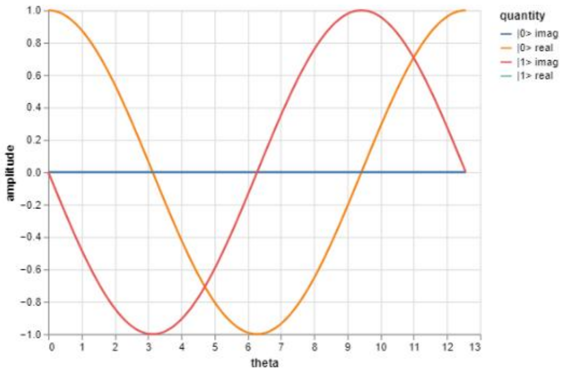
```
dev = qml.device("default.qubit", wires=1)

@qml.qnode(dev)
def apply_rx(theta, state):
    """Apply an RX gate with an angle of theta to a particular basis state.

    Args:
        theta (float): A rotation angle.
        state (int): Either 0 or 1. If 1, initialize the qubit to state |1>
            before applying other operations.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    if state == 1:
        qml.PauliX(wires=0)

    ##################
    # APPLY RX(theta) AND RETURN THE STATE
    qml.RX(theta,wires=0)
    ##################

    # APPLY RX(theta) AND RETURN THE STATE

    return qml.state()

# Code for plotting
angles = np.linspace(0, 4 * np.pi, 200)
output_states = np.array([apply_rx(t, 0) for t in angles])

plot = plotter(angles, output_states)
```

```
# Code for plotting
angles = np.linspace(0, 4 * np.pi, 200)
output_states = np.array([apply_rx(t, 0) for t in angles])

plot = plotter(angles, output_states)
```

Reset Code Submit

Correct!

User output



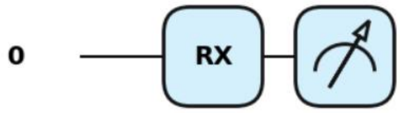
Qiskit Program:

```
import numpy as np
import random
from qiskit.quantum_info import Statevector
import pennylane as qml
import matplotlib.pyplot as plt

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

@qml.qnode(dev)
def apply_rx(theta, state):
    """Apply an RX gate with an angle of theta to a particular basis state.

    Args:
        theta (float): A rotation angle.
        state (int): Either 0 or 1. If 1, initialize the qubit to state |1>
            before applying other operations.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    if state == 1:
        qml.PauliX(wires=0)

    ##################
    # APPLY RX(theta) AND RETURN THE STATE
    qml.RX(theta,wires=0)
    ##################

    return qml.state()

# Code for plotting
```

```
angles = np.linspace(0, 4 * np.pi, 200)
output_states = np.array([apply_rx(t, 0) for t in angles])
```

O/P:

```
circuit = qml.QNode(apply_rx, dev)
qml.drawer.use_style("pennylane")
result = qml.draw_mpl(circuit)(angles,0)
plt.show()
```



Codercise I.6.3 — Plotting RY

Open related theory

Repeat the above exercise, but using `qml.RY`. From the amplitudes you obtain for $RY(\theta)|0\rangle$, can you start deducing the matrix form of RY?

Solution:

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

@qml.qnode(dev)
def apply_ry(theta, state):
    """Apply an RY gate with an angle of theta to a particular basis state.

    Args:
        theta (float): A rotation angle.
        state (int): Either 0 or 1. If 1, initialize the qubit to state |1>
            before applying other operations.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    if state == 1:
        qml.PauliX(wires=0)

    ##################
    qml.RY(theta,wires=0)
    ##################
    # APPLY RY(theta) AND RETURN THE STATE
    return qml.state()

# Code for plotting
angles = np.linspace(0, 4 * np.pi, 200)
output_states = np.array([apply_ry(t, 0) for t in angles])

plot = plotter(angles, output_states)
```

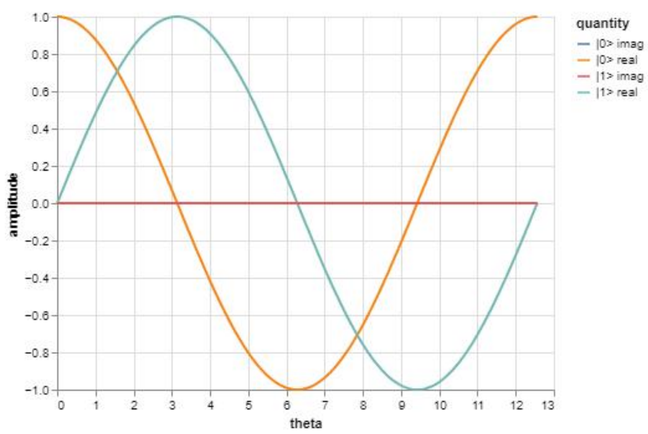
```
    ##################

    # APPLY RY(theta) AND RETURN THE STATE

    return qml.state()


# Code for plotting
angles = np.linspace(0, 4 * np.pi, 200)
output_states = np.array([apply_ry(t, 0) for t in angles])

plot = plotter(angles, output_states)
```

Reset Code

Submit

Correct!

User output



Qiskit Program:

```
import numpy as np
import random
from qiskit.quantum_info import Statevector
import pennylane as qml
import matplotlib.pyplot as plt

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

@qml.qnode(dev)
def apply_ry(theta, state):
    """Apply an RX gate with an angle of theta to a particular basis state.

    Args:
        theta (float): A rotation angle.
        state (int): Either 0 or 1. If 1, initialize the qubit to state |1>
            before applying other operations.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    if state == 1:
        qml.PauliX(wires=0)
```

```
    ##################
    # APPLY RX(theta) AND RETURN THE STATE
    qml.RY(theta,wires=0)
    ##################

    return qml.state()

# Code for plotting
angles = np.linspace(0, 4 * np.pi, 200)
output_states = np.array([apply_ry(t, 0) for t in angles])

#plot = plotter(angles, output_states)
```

O/P:

```
circuit = qml.QNode(apply_ry, dev)
qml.drawer.use_style("pennylane")
result = qml.draw_mpl(circuit)(angles,0)
plt.show()
```