Exercise - Multiple Systems

December 11, 2024

1 Multiple Systems

In the previous lesson, we learned about Qiskit's Statevector and Operator classes, and used them to simulate quantum systems. In this section, we'll use them to explore the behavior of multiple systems. We'll start by importing these classes, as well as the square root function from NumPy .

[1]:
```
%pip install qiskit[visualization]
```

```
Requirement already satisfied: qiskit[visualization] in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages
(1.3.0)
Requirement already satisfied: rustworkx>=0.15.0 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit[visualization]) (0.15.1)
Requirement already satisfied: numpy<3,>=1.17 in /opt/conda/lib/python3.11/site-
packages (from qiskit[visualization]) (1.26.4)
Requirement already satisfied: scipy>=1.5 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit[visualization]) (1.14.1)
Requirement already satisfied: sympy>=1.3 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit[visualization]) (1.13.3)
Requirement already satisfied: dill>=0.3 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit[visualization]) (0.3.9)
Requirement already satisfied: python-dateutil>=2.8.0 in
/opt/conda/lib/python3.11/site-packages (from qiskit[visualization]) (2.9.0)
Requirement already satisfied: stevedore>=3.0.0 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit[visualization]) (5.4.0)
Requirement already satisfied: typing-extensions in
/opt/conda/lib/python3.11/site-packages (from qiskit[visualization]) (4.12.2)
Requirement already satisfied: symengine<0.14,>=0.11 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit[visualization]) (0.13.0)
Requirement already satisfied: matplotlib>=3.3 in
/opt/conda/lib/python3.11/site-packages (from qiskit[visualization]) (3.9.3)
Requirement already satisfied: pydot in
```

```
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit[visualization]) (3.0.3)
Requirement already satisfied: Pillow>=4.2.1 in /opt/conda/lib/python3.11/site-
packages (from qiskit[visualization]) (11.0.0)
Requirement already satisfied: pylatexenc>=1.4 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit[visualization]) (2.10)
Requirement already satisfied: seaborn>=0.9.0 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit[visualization]) (0.13.2)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.11/site-packages (from
matplotlib>=3.3->qiskit[visualization]) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.11/site-
packages (from matplotlib>=3.3->qiskit[visualization]) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.11/site-packages (from
matplotlib>=3.3->qiskit[visualization]) (4.55.2)
Requirement already satisfied: kiwisolver>=1.3.1 in
/opt/conda/lib/python3.11/site-packages (from
matplotlib>=3.3->qiskit[visualization]) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.11/site-packages (from
matplotlib>=3.3->qiskit[visualization]) (24.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/conda/lib/python3.11/site-packages (from
matplotlib>=3.3->qiskit[visualization]) (3.2.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.11/site-
packages (from python-dateutil>=2.8.0->qiskit[visualization]) (1.16.0)
Requirement already satisfied: pandas>=1.2 in /opt/conda/lib/python3.11/site-
packages (from seaborn>=0.9.0->qiskit[visualization]) (2.2.3)
Requirement already satisfied: pbr>=2.0.0 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
stevedore>=3.0.0->qiskit[visualization]) (6.1.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
sympy>=1.3->qiskit[visualization]) (1.3.0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.11/site-
packages (from pandas>=1.2->seaborn>=0.9.0->qiskit[visualization]) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.11/site-
packages (from pandas>=1.2->seaborn>=0.9.0->qiskit[visualization]) (2024.2)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
from qiskit.quantum_info import Statevector, Operator
from numpy import sqrt
```

2 Tensor products

The Statevector class has a tensor method which returns the tensor product of itself and another Statevector . For example, below we create two state vectors representing 0 and 1 , and use the tensor method to create a new vector, 0 1 .

[4]:
```
# Replace ?
zero, one = Statevector.from_label("0"), Statevector.from_label("1")
zero.tensor(one).draw("latex")
```

[4]:

$$|01\rangle$$

In another example below, we create state vectors representing the + and (0 +i 1)/sqrt(2) states, and combine them to create a new state vector. We'll assign this new vector to the variable psi .

[5]:
```
# Replace ?
plus = Statevector.from_label("+")
i_state = Statevector([1 / sqrt(2), 1j / sqrt(2)])
psi = plus.tensor(i_state)

psi.draw("latex")
```

[5]:

$$\frac{1}{2}|00\rangle + \frac{i}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{i}{2}|11\rangle$$

The Operator class also has a tensor method. In the example below, we create the X and I gates and display their tensor product.

[6]:
```
# Replace ?
X = Operator([[0, 1], [1, 0]])
I = Operator([[1, 0], [0, 1]])

X.tensor(I)
```

```
Operator([[0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j],
          [0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j],
          [1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j],
          [0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j]],
         input_dims=(2, 2), output_dims=(2, 2))
```

We can then treat these compound states and operations as we did single systems in the previous lesson. For example, in the cell below we calculate (I X) for the state psi we defined above. (The ^ operator tensors matrices together.)

[7]:
```
# Replace ?
psi.evolve(X ^ I).draw("latex")
```

[7]:

$$\frac{1}{2}|00\rangle + \frac{i}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{i}{2}|11\rangle$$

Below, we create a CX operator and calculate CX .

[9]:
```
# Replace ?
CX = Operator(
    [
        [1, 0, 0, 0],
        [0, 0, 0, 1],
        [0, 0, 1, 0],
        [0, 1, 0, 0],
    ]
)

psi.evolve(CX).draw("latex")
```

[9]:

$$\frac{1}{2}|00\rangle + \frac{i}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{i}{2}|11\rangle$$

3 Partial measurements

In the previous part, we used the measure method to simulate a measurement of the quantum state vector.

This method returns two items: the simulated measurement result, and the new Statevector given this measurement.

By default, `measure` measures all qubits in the state vector, but we can provide a list of integers to only measure the qubits at those indices. To demonstrate, the cell below creates the state W = 1/sqrt(3) (001 + 010 + 100).

Note that Qiskit is primarily designed for use with qubit-based digital quantum computers. As such, Statevector will try to interpret any vector with 2**n elements as a system of n qubits.

You can override this by passing a dims argument to the constructor. For example, dims=(4,2) would tell Qiskit the system has one four-level system, and one two-level system (qubit).)

[10]:
```
W = Statevector([0, 1, 1, 0, 1, 0, 0, 0] / sqrt(3))
W.draw("latex")
```

[10]:

$$\frac{\sqrt{3}}{3}|001\rangle + \frac{\sqrt{3}}{3}|010\rangle + \frac{\sqrt{3}}{3}|100\rangle$$

The cell below simulates a measurement on the rightmost qubit (which has index 0). The other two qubits are not measured.

[11]:
```
result, new_sv = W.measure([0])  # measure qubit 0
print(f"Measured: {result}\nState after measurement:")
```

```
new_sv.draw("latex")
```

```
Measured: 0
State after measurement:
```

[11]:

$$\frac{\sqrt{2}}{2}|010\rangle + \frac{\sqrt{2}}{2}|100\rangle$$

Try running the cell a few times to see different results. Notice that measuring a 1 means that we know both the other qubits are 0 , but measuring a 0 means the remaining two qubits are in the state 1/sqrt(2) x (01 + 10).

Repeat this in the cells below!

The cells below simulates another measurement on the rightmost qubit (which has index 0). The other two qubits are not measured.

[12]:
```
# Replace ?
W = Statevector([0, 1, 1, 0, 1, 0, 0, 0] / sqrt(3))
W.draw("latex")
```

[12]:

$$\frac{\sqrt{3}}{3}|001\rangle + \frac{\sqrt{3}}{3}|010\rangle + \frac{\sqrt{3}}{3}|100\rangle$$

[13]:
```
# Replace?
result, new_sv = W.measure([0])  # measure qubit 0
print(f"Measured: {result}\nState after measurement:")
new_sv.draw("latex")
```

```
Measured: 0
State after measurement:
```

[13]:

$$\frac{\sqrt{2}}{2}|010\rangle + \frac{\sqrt{2}}{2}|100\rangle$$

4 End of Notebook