2.3.3 Addendum: $\pi/8$-Gate

In previous sections we discussed the Phase gate (also known as the S-gate or, in some literature, as the P-gate):

$$S = P = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/2} \end{bmatrix},$$

which represents a rotation by $\pi/2$ or 90°, given Euler's formula:

$$e^{i\pi/2} = \cos(\pi/2) + i\sin(\pi/2) = i$$

We can see that the S-gate is the square root of the Z-gate:

$$Z = S^2 = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/2*2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

We also learned about the T-gate, which is the root of S. It represents a rotation by half of the S-gate, which is a rotation by 45°:

$$T = \sqrt{S} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$$

The T-gate is sometimes called the $\pi/8$ gate, which sees counter-intuitive, given that the gate has a factor of $\pi/4$ in it! The name comes from the fact that we can make the gate more symmetric by pulling out a factor, as in:

$$T \;=\; \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \;=\; e^{i\pi/8} \begin{bmatrix} e^{-i\pi/8} & 0 \\ 0 & e^{i\pi/8} \end{bmatrix}$$

2.3.4 Addendum: U-Gate

Physical quantum computers may implement other, "non-standard" types of gates. IBM machines specifically provide the general *U-gate*:

$$U(\theta, \phi, \lambda) = \begin{bmatrix} \cos(\theta/2) & -e^{-i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}\cos(\theta/2) \end{bmatrix}.$$

Note that in section 8.4.8 of the book, gate (u_3) has an error in the top right element (this has also been added to the book's errata):

$$u_3(\theta, \phi, \lambda) = \begin{bmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}\cos(\theta/2) \end{bmatrix}`.$$

The U-gate is quite versatile, as it can be used to construct many other standard gates [1]. For example:

$$\begin{aligned} U(\theta = \frac{\pi}{2}, \phi = 0, \lambda = \pi) &= \begin{bmatrix} \cos(\theta/2) & -e^{-i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}\cos(\theta/2) \end{bmatrix} \\ &= \begin{bmatrix} \cos(\pi/4) & -(-1)\sin(\pi/4) \\ 1\sin(\pi/4) & -1\cos(\pi/4) \end{bmatrix}. \end{aligned}$$

[1] https://qiskit.org/textbook/ch-states/single-qubit-gates.html, Section 7

With:

$$\cos\frac{\pi}{4} = \sin\frac{\pi}{4} = \frac{1}{\sqrt{2}},$$

we are able to construct a Hadamard gate:

$$\Rightarrow U(\frac{\pi}{2}, 0, \pi) = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} = H$$

Another example is how to construct the flexible phase gate U_1 (with which we can generate the P-gate, S-gate, and T-gate):

$$\begin{aligned}\Rightarrow U(0,0,\lambda) &= \begin{bmatrix}\cos(\theta/2) & -e^{-i\lambda}\sin(\theta/2)\\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}\cos(\theta/2)\end{bmatrix}\\ &= \begin{bmatrix}1 & 0\\ 0 & e^{i\lambda}\end{bmatrix}\\ &= U_1.\end{aligned}$$

Specifically, to generate a Z-gate:

$$U(0,0,\pi) = \begin{bmatrix}1 & 0\\ 0 & e^{i\pi}\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} = Z$$

Can we make an X-gate? To achieve this, we need $\cos(\theta/2) = 0$, which we can achieve by setting $\theta = \pi$. With this, the $\sin(\cdot)$ terms become 1:

$$U(\pi,\phi,\lambda) = \begin{bmatrix}0 & -e^{-i\lambda}\\ e^{i\phi} & 0\end{bmatrix} \tag{2.1}$$

The lower left term must be 1, hence $\phi = 0$. The upper right term must be 1 as well, hence $\lambda = \pi$. We arrive at:

$$U(\pi,0,\pi) = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} = X$$

From Equation 2.1 we can also derive the Pauli Y-gate as:

$$U(\pi,\frac{\pi}{2},-\frac{\pi}{2}) = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix} = Y$$

Finally, the identity gate is easy to construct as

$$U(0,0,0) = I$$

.

Let's verify this in code. Similar to all the other standard gates, we add this constructor to `lib/ops.py`:

```
# IBM's general U-gate.
def U(theta: float, phi: float, lam: float, d: int = 1) -> Operator:
  return Operator(np.array(
     [(np.cos(theta/2),
```

```
    -cmath.exp(-1j*lam)*np.sin(theta/2)),
    (cmath.exp(1j*phi)*np.sin(theta/2),
    cmath.exp(1j*(phi+lam))*np.cos(theta/2))])).kpow(d)
```

We add a few tests to verify that our constructions were correct:

```
def test_u(self):
  val = random.random()
  self.assertTrue(np.allclose(ops.U(0, 0, val),
                              ops.U1(val)))
  self.assertTrue(np.allclose(ops.U(np.pi/2, 0, np.pi),
                              ops.Hadamard()))
  self.assertTrue(np.allclose(ops.U(0, 0, 0),
                              ops.Identity()))
  self.assertTrue(np.allclose(ops.U(np.pi, 0, np.pi),
                              ops.PauliX()))
  self.assertTrue(np.allclose(ops.U(np.pi, np.pi/2, -np.pi/2),
                              ops.PauliY()))
  self.assertTrue(np.allclose(ops.U(0, 0, np.pi),
                              ops.PauliZ()))
```