Assessments

Chapter 1, Foundations of Quantum Computing

(1.1) The probability of measuring 0 if the state of a qubit is $\sqrt{1/2}\,|0\rangle + \sqrt{1/2}\,|1\rangle$ is exactly

$$\left|\sqrt{1/2}\right|^2 = 1/2.$$

In the same way, the probability of measuring 1 is also 1/2. If the state of the qubit is $\sqrt{1/3}\,|0\rangle + \sqrt{2/3}\,|1\rangle$, the probability of measuring 0 is

$$\left|\sqrt{1/3}\right|^2 = 1/3$$

and the probability of measuring 1 is

$$\left|\sqrt{2/3}\right|^2 = 2/3.$$

Finally, if the qubit state is $\sqrt{1/2}\,|0\rangle - \sqrt{1/2}\,|1\rangle$, the probability of measuring 0 is

$$\left|\sqrt{1/2}\right|^2 = 1/2$$

and the probability of measuring 1 is

$$\left|-\sqrt{1/2}\right|^2 = 1/2.$$

(1.2) The inner product of $\sqrt{1/2}\,|0\rangle + \sqrt{1/2}\,|1\rangle$ and $\sqrt{1/3}\,|0\rangle + \sqrt{2/3}\,|1\rangle$ is

$$\sqrt{1/2}\sqrt{1/3} + \sqrt{1/2}\sqrt{2/3} = \sqrt{1/6} + \sqrt{1/3}.$$

The inner product of $\sqrt{1/2}\,|0\rangle + \sqrt{1/2}\,|1\rangle$ and $\sqrt{1/2}\,|0\rangle - \sqrt{1/2}\,|1\rangle$ is

$$\sqrt{1/2}\sqrt{1/2} - \sqrt{1/2}\sqrt{1/2} = 0.$$

(1.3) The adjoint of X is X itself and it holds that $XX = I$. Hence, X is unitary and its inverse is X itself. The operation X takes $a\,|0\rangle + b\,|1\rangle$ to $b\,|0\rangle + a\,|1\rangle$.

(1.4) The adjoint of H is H itself and it holds that $HH = I$. Hence, H is unitary and its inverse is H itself. The operation H takes $|+\rangle$ to $|0\rangle$ and $|-\rangle$ to $|1\rangle$. Finally, it holds that $X\,|+\rangle = |+\rangle$ and that $X\,|-\rangle = -\,|-\rangle$.

(1.5) It holds that

$$Z\,|0\rangle = HXH\,|0\rangle = HX\,|+\rangle = H\,|+\rangle = |0\rangle$$

and that

$$Z\,|1\rangle = HXH\,|1\rangle = HX\,|-\rangle = -H\,|-\rangle = -\,|1\rangle\,.$$

It also holds that

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}\begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

(1.6) Since $e^{i\frac{\pi}{4}}e^{i\frac{\pi}{4}} = e^{i\frac{\pi}{2}}$, it is apparent that $T^2 = S$. Also, we have $e^{i\frac{\pi}{2}}e^{i\frac{\pi}{2}} = e^{i\pi} = -1$ by Euler's identity, so $S^2 = Z$. As a consequence, $SS^3 = S^2S^2 = ZZ = I$, so $S^\dagger = S^3$. Also, $TT^7 = T^2T^2T^2T^2 = S^4 = I$, and it follows that $T^\dagger = T^7$.

(1.7) By the definition of R_X we have that

$$R_X(\pi) = \begin{pmatrix} \cos\frac{\pi}{2} & -i\sin\frac{\pi}{2} \\ -i\sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{pmatrix} = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix} = -iX.$$

Analogously,

$$R_Y(\pi) = \begin{pmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = -iY$$

and

$$R_Z(\pi) = \begin{pmatrix} e^{-i\frac{\pi}{2}} & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix} = \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix} = -iZ.$$

Also,

$$R_Z\left(\frac{\pi}{2}\right) = \begin{pmatrix} e^{-i\frac{\pi}{4}} & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} = e^{-i\frac{\pi}{4}}S$$

and

$$R_Z\left(\frac{\pi}{4}\right) = \begin{pmatrix} e^{-i\frac{\pi}{8}} & 0 \\ 0 & e^{i\frac{\pi}{8}} \end{pmatrix} = e^{-i\frac{\pi}{8}}T.$$

(1.8) From the definition of $U(\theta, \varphi, \lambda)$, we have that

$$U(\theta,\varphi,\lambda)U(\theta,\varphi,\lambda)^\dagger = \begin{pmatrix} \cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2} \\ e^{i\varphi}\sin\frac{\theta}{2} & e^{i(\varphi+\lambda)}\cos\frac{\theta}{2} \end{pmatrix} \begin{pmatrix} \cos\frac{\theta}{2} & e^{-i\varphi}\sin\frac{\theta}{2} \\ -e^{-i\lambda}\sin\frac{\theta}{2} & e^{-i(\varphi+\lambda)}\cos\frac{\theta}{2} \end{pmatrix} = I$$

and, analogously, $U(\theta,\varphi,\lambda)^\dagger U(\theta,\varphi,\lambda) = I$. Then, $U(\theta,\varphi,\lambda)$ is unitary.

Also, we get that

$$U(\theta, -\pi/2, \pi/2) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} = R_X(\theta).$$

Analogously, it holds that

$$U(\theta, 0, 0) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} = R_Y(\theta)$$

and that

$$U(0, 0, \theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} = e^{i\frac{\theta}{2}} R_Z(\theta).$$

(1.9) Since $\theta = 2\arccos\sqrt{p}$, it holds that the state before measurement is

$$\cos\frac{\theta}{2}\left|0\right\rangle + \sin\frac{\theta}{2}\left|1\right\rangle = \sqrt{p}\left|0\right\rangle + \sqrt{1-p}\left|1\right\rangle.$$

As a consequence, the probability of measuring 0 is p and the probability of measuring 1 is $1 - p$.

(1.10) The probability of obtaining 1 will be $|a_{10}|^2 + |a_{11}|^2$. Upon that measurement result, the state will collapse to

$$\frac{a_{10}\left|10\right\rangle + a_{11}\left|11\right\rangle}{\sqrt{|a_{10}|^2 + |a_{11}|^2}}.$$

(1.11) It holds that

$$(U_1 \otimes U_2)(U_1^\dagger \otimes U_2^\dagger) = (U_1U_1^\dagger) \otimes (U_2U_2^\dagger) = I \otimes I.$$

Analogously, $(U_1^\dagger \otimes U_2^\dagger)(U_1 \otimes U_2) = I \otimes I$. Hence, the inverse of $U_1 \otimes U_2$ is $U_1^\dagger \otimes U_2^\dagger$.

Also, from the definition of tensor product of two matrices we get that, for every matrix A and B (even in they are non-unitary), it holds that

$$A^\dagger \otimes B^\dagger = \begin{pmatrix} a_{11}^* & a_{21}^* \\ a_{12}^* & a_{22}^* \end{pmatrix} \otimes \begin{pmatrix} b_{11}^* & b_{21}^* \\ b_{12}^* & b_{22}^* \end{pmatrix} = \begin{pmatrix} a_{11}^* \begin{pmatrix} b_{11}^* & b_{21}^* \\ b_{12}^* & b_{22}^* \end{pmatrix} & a_{21}^* \begin{pmatrix} b_{11}^* & b_{21}^* \\ b_{12}^* & b_{22}^* \end{pmatrix} \\ a_{12}^* \begin{pmatrix} b_{11}^* & b_{21}^* \\ b_{12}^* & b_{22}^* \end{pmatrix} & a_{22}^* \begin{pmatrix} b_{11}^* & b_{21}^* \\ b_{12}^* & b_{22}^* \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}^*b_{11}^* & a_{11}^*b_{21}^* & a_{21}^*b_{11}^* & a_{21}^*b_{21}^* \\ a_{11}^*b_{12}^* & a_{11}^*b_{22}^* & a_{21}^*b_{12}^* & a_{21}^*b_{22}^* \\ a_{12}^*b_{11}^* & a_{12}^*b_{21}^* & a_{22}^*b_{11}^* & a_{22}^*b_{21}^* \\ a_{12}^*b_{12}^* & a_{12}^*b_{22}^* & a_{22}^*b_{12}^* & a_{22}^*b_{22}^* \end{pmatrix} = (A \otimes B)^\dagger.$$

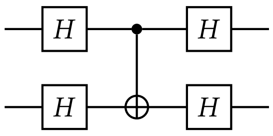
(1.12) The matrix for $X \otimes X$ is

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

The matrix for $H \otimes I$ is

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}.$$

(1.13) In the circuit



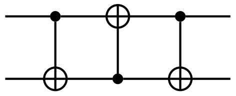
the states $|00\rangle$ and $|10\rangle$ are left unchanged, while $|01\rangle$ and $|11\rangle$ are mapped to each other. This is exactly the action of a CNOT gate whose control is the bottom qubit and whose target is the top one.

The matrix for the circuit is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix},$$

which is exactly the matrix for the CNOT gate from bottom qubit to top qubit.

On the other hand, the circuit



leaves $|00\rangle$ and $|11\rangle$ unchanged, while it maps $|01\rangle$ and $|10\rangle$ to each other. This is exactly the action of a SWAP gate.

Alternatively, the matrix for the circuit is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

which is, again, the matrix of the SWAP gate.

(1.14) The state $\sqrt{1/3}(|00\rangle + |01\rangle + |11\rangle)$ is indeed entangled. However, $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$ is a product state because it can be written as $|+\rangle\,|+\rangle$.

(1.15) If the matrix of U is $(u_{ij})_{i,j=1}^{2}$, then $|00\rangle$ and $|01\rangle$ are taken to themselves by CU. What is more, $|10\rangle$ is taken to $|1\rangle\,(u_{11}\,|0\rangle + u_{21}\,|1\rangle)$ and $|11\rangle$ is taken to $|1\rangle\,(u_{12}\,|0\rangle + u_{22}\,|1\rangle)$.

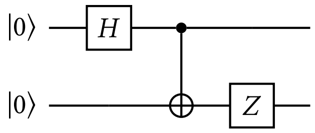
Hence, the matrix of CU is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{11} & u_{12} \\ 0 & 0 & u_{21} & u_{22} \end{pmatrix}.$$

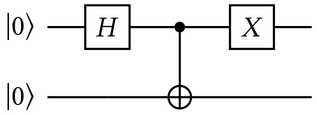
The adjoint of CU is $CU^\dagger$ and it holds that $CUCU^\dagger = CU^\dagger CU = I$. Hence, CU is unitary.

(1.16) The equivalence follows directly from the fact that $HXH = Z$.

(1.17) We can prepare $\sqrt{1/2}\,(|00\rangle - |11\rangle)$ with the following circuit:

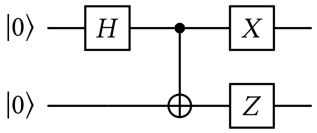


We can use the circuit



to prepare $\sqrt{1/2}(|10\rangle + |01\rangle)$.

Finally, the circuit



can be used to obtain $\sqrt{1/2}(|10\rangle - |01\rangle)$.

Notice that, to prepare these states, we are only using tensor product gates appended to the circuit that we used to obtain the original Bell state $\sqrt{1/2}(|00\rangle + |11\rangle)$. For instance, we have that

$$\sqrt{1/2}(|10\rangle - |01\rangle) = (X \otimes Z)\sqrt{1/2}(|00\rangle + |11\rangle)$$

and, then, it also holds that

$$(X \otimes Z)\sqrt{1/2}(|10\rangle - |01\rangle) = \sqrt{1/2}(|00\rangle + |11\rangle).$$

If $\sqrt{1/2}(|10\rangle - |01\rangle)$ were a product state, then $\sqrt{1/2}(|00\rangle + |11\rangle)$ would also be a product state. But that is impossible, because we know that $\sqrt{1/2}(|00\rangle + |11\rangle)$ is entangled.

(1.18) We can prove it by induction. We know that the result is true for $n = 1$. Now, assume that it is true for $n > 1$ and consider a basis state $|\psi\rangle$ of $n + 1$ qubits. If $|\psi\rangle = |0\rangle |\psi'\rangle$, then the column vector for $|\psi\rangle$ will start with the elements of the column vector of $|\psi'\rangle$ and then it will have 2^n zeroes. But the column vector for $|\psi'\rangle$ is, by the induction hypothesis, exactly of the form that we are interested in. It follows that $|\psi\rangle$ also has the desired structure. The case when $|\psi\rangle = |1\rangle |\psi'\rangle$ is analogous.

On the other hand, since every n-qubit state can be written as a normalized linear combination of basis states, it follows that its vector representation is a unit length column vector with 2^n coordinates.

(1.19) If we measure the j-th qubit of a generic multi-qubit state, the probability of obtaining 1 is given by

$$\sum_{l\in J_1} |a_l|^2,$$

where J_1 is the set of numbers whose j-th bit is 1. The state after the collapse will be

$$\frac{\sum_{l\in J_1} a_l |l\rangle}{\sqrt{\sum_{l\in J_1} |a_i|^2}}.$$

(1.20) The probability of getting 0 when we measure the second qubit of $(1/2)|100\rangle + (1/2)|010\rangle + \sqrt{1/2}\,|001\rangle$ is

$$\left|\frac{1}{2}\right|^2 + \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{4} + \frac{1}{2} = \frac{3}{4}.$$

The result after measuring the second qubit and obtaining 0 would be

$$\frac{1}{\sqrt{3}}\left|100\right\rangle + \frac{\sqrt{2}}{\sqrt{3}}\left|001\right\rangle.$$

(1.21) Let's denote $x = x_1 \dots x_n$ and $y = y_1 \dots y_n$, where x_i is the i-th bit of x and y_i is the i-th bit of y. Then, it holds that

$$\left\langle y|x\right\rangle = \left\langle y_1|x_1\right\rangle \dots \left\langle y_n|x_n\right\rangle.$$

As a consequence, $\langle y|x\rangle = 1$ if $x = y$ and $\langle y|x\rangle = 0$ if $x \neq y$. From this, it follows that the elements in $\{|x\rangle\}_{x\in\{0,1\}^n}$ are orthonormal. Since the cardinality of this set is 2^n, which is the dimension of n-qubit states, we can conclude that the set forms a basis.

(1.22) It holds that

$$\begin{aligned}
&\frac{1}{\sqrt{2}}\left(\langle 000| + \langle 111|\right)\frac{1}{2}\left(|000\rangle + |011\rangle + |101\rangle + |110\rangle\right) \\
&\quad= \frac{1}{2\sqrt{2}}(\langle 000|000\rangle + \langle 000|011\rangle + \langle 000|101\rangle + \langle 000|110\rangle + \\
&\qquad \langle 111|000\rangle + \langle 111|011\rangle + \langle 111|101\rangle + \langle 111|110\rangle) \\
&\quad= \frac{1}{2\sqrt{2}},
\end{aligned}$$

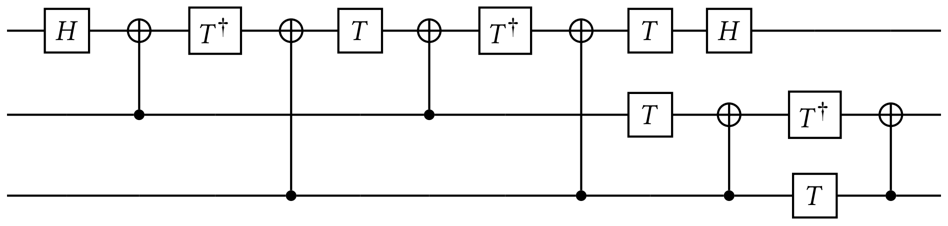
because all the inner products are 0 except $\langle 000|000\rangle$, which is 1.

(1.23) From its action of the basis states, we deduce that the matrix for the CCNOT gate is:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

It holds that this matrix is its own adjoint and that its square is the identity. As a consequence, the matrix is unitary.

(1.24) The circuit



leaves all the states but $|011\rangle$ and $|111\rangle$ unchanged. It also interchanges $|011\rangle$ and $|111\rangle$. That is exactly the action of a CCNOT gate with target on the top qubit.

Chapter 2, The Tools of the Trade in Quantum Computing

(2.1) We already gave you the solution in *Appendix D, Installing the Tools.*

(2.2) In order to construct the circuit in *Figure 2.3b*, you would have to execute the following piece of code:

```
from qiskit import *
import numpy as np
```

```
qc = QuantumCircuit(2)

qc.z(0)
qc.y(1)
qc.cry(np.pi/2, 0, 1)
qc.u(np.pi/4, np.pi, 0, 0)
qc.rz(np.pi/4,1)
```

If you want to visualize the circuit, of course, you can use `qc.draw("mpl")`.

(2.3) You can check IBM's own implementation (`https://github.com/Qiskit/qiskit-terra/blob/5ccf3a41cb10742ae2158b6ee9d13bbb05f64f36/qiskit/circuit/quantumcircuit.py#L2205`) of the method and compare it to your own!

They take some additional steps that we have not considered, such as adding **barriers** in the circuit, but you can ignore those details.

(2.4) You already have the solution in *Appendix D, Installing the Tools.*

(2.5) We have already seen how to construct these circuits in Qiskit. In order to construct them in PennyLane, we would need to run the following piece of code:

```
import pennylane as qml
import numpy as np
dev = qml.device('default.qubit', wires = 2)
@qml.qnode(dev)
def qcircA():
    qml.PauliX(wires = 0)
    qml.RX(np.pi/4, wires = 1)
    qml.CNOT(wires = [0,1])
    qml.U3(np.pi/3, 0, np.pi, wires = 0)
    return qml.state()


@qml.qnode(dev)
def qcircB():
    qml.PauliZ(wires = 0)
    qml.PauliY(wires = 1)
    qml.CRY(np.pi/2, wires = [0,1])
    qml.U3(np.pi/4, np.pi, 0, wires = 0)
    qml.RZ(np.pi/4, wires = 1)
    return qml.state()
```

If we run this, for instance, for circuit B by executing `print(qcircB()`, we get the following state vector:

```
tensor([ 0.        +0.j        , -0.35355339+0.85355339j,
         0.        +0.j        ,  0.14644661-0.35355339j],
         requires_grad=True)
```

On the other hand, if we simulate that very same circuit with Qiskit, we get this output:

```
Statevector([-5.65831421e-17-3.20736464e-17j,
              2.34375049e-17+1.32853393e-17j,
             -3.53553391e-01+8.53553391e-01j,
              1.46446609e-01-3.53553391e-01j],
            dims=(2, 2))
```

Notice that this is the same result that we got with PennyLane. We first have to take into account the fact that the first two entries are — computationally speaking — zero. And then we have to draw our attention to how Qiskit, following its own conventions, gives us the amplitudes of the basis states in the following order: $|00\rangle$, $|10\rangle$, $|01\rangle$, and $|11\rangle$.

Chapter 3, Working with Quadratic Unconstrained Binary Optimization Problems

(3.1) We can put vertices 0, 1, and 4 in one group, and vertices 2 and 3 in the other. Then, five edges belong to the cut, namely $(0, 2), (1, 2), (1, 3), (2, 4)$, and $(3, 4)$.

(3.2) The optimization problem for the Max-Cut of the graph in *Figure 3.3* is

$$\begin{array}{ll} \text{Minimize} & z_0z_1 + z_0z_2 + z_1z_2 + z_1z_4 + z_2z_3 + z_3z_4 + z_3z_5 + z_4z_5 \\ \text{subject to} & z_j \in \{-1, 1\}, \qquad j = 0, \ldots, 5. \end{array}$$

The value of the cut given by $z_0 = z_1 = z_2 = 1$ and $z_3 = z_4 = z_5 = -1$ is 4. This cut is not optimal, because, for instance, $z_0 = z_1 = z_2 = z_5 = 1$ and $z_3 = z_4 = -1$ achieves a lower value.

(3.3) It holds that $\langle 010| (Z_0Z_1 + Z_0Z_2) |010\rangle = 0$ and that $\langle 100| (Z_0Z_1 + Z_0Z_2) |100\rangle = -2$. This latter value is the minimum possible, because we only have two edges in our graph.

(3.4) We can compute the required expectation values with the following code:

```
from qiskit.quantum_info import Pauli
from qiskit.opflow.primitive_ops import PauliOp
from qiskit.quantum_info import Statevector
H_cut = PauliOp(Pauli("ZZI")) + PauliOp(Pauli("ZIZ"))
for x in range(8): # We consider x=0,1...7
    psi = Statevector.from_int(x, dims = 8)
    print("The expectation value of |",x,">", "is",
        psi.expectation_value(H_cut))
```

If we run it, we obtain the following output:

```
The expectation value of | 0 > is (2+0j)
The expectation value of | 1 > is 0j
The expectation value of | 2 > is 0j
The expectation value of | 3 > is (-2+0j)
The expectation value of | 4 > is (-2+0j)
The expectation value of | 5 > is 0j
The expectation value of | 6 > is 0j
The expectation value of | 7 > is (2+0j)
```

Thus, we can see that there are two states that obtain the optimal value and both correspond to the cut in which 0 is in one group and 1 and 2 in the other.

(3.5) The QUBO problem would be

$$\begin{aligned}\text{Minimize}\quad & x_0^2 - 4x_0x_1 + 6x_0x_2 - 8x_0x_3 + 4x_1^2 - 12x_1x_2 + 16x_1x_3 + 9x_2^2\\ & - 24x_2x_3 + 16x_3^2\\ \text{subject to}\quad & x_j \in \{0,1\}, \qquad j = 0,1,2,3.\end{aligned}$$

The equivalent Ising ground state problem would be

$$\begin{aligned}\text{Minimize}\quad & -z_0z_1 + \frac{3z_0z_2}{2} - 2z_0z_3 + z_0 - 3z_1z_2 + 4z_1z_3 - 2z_1 - 6z_2z_3 + 3z_2 - 4z_3\\ \text{subject to}\quad & z_j \in \{1,-1\}, \qquad j = 0,1,2,3,\end{aligned}$$

where we have dropped the independent term $\frac{17}{2}$.

(3.6) The binary linear program would be

$$\begin{aligned}\text{Minimize}\quad & -3x_0 - x_1 - 7x_2 - 7x_3\\ \text{subject to}\quad & 2x_0 + x_1 + 5x_2 + 4x_3 \le 8,\\ & x_j \in \{0,1\}, \qquad j = 0,1,2,3.\end{aligned}$$

(3.7) The QUBO problem is

$$\begin{aligned}
\text{Minimize} \quad & (x_{00}+x_{01}-1)^2+(x_{10}+x_{11}-1)^2+(x_{20}+x_{21}-1)^2+(x_{30}+x_{31}-1)^2 \\
& +x_{00}x_{10}+x_{01}x_{11}+x_{00}x_{20}+x_{01}x_{21}+x_{10}x_{30}+x_{11}x_{31}+x_{20}x_{30}+x_{21}x_{31} \\
\text{subject to} \quad & x_{jk}\in\{0,1\}, \qquad j=0,1,2,3, k=0,1.
\end{aligned}$$

(3.8) The expression for the route cost is

$$\begin{aligned}
&+2x_{00}x_{11}+x_{00}x_{21}+3x_{00}x_{31}+2x_{10}x_{01}+4x_{10}x_{21}+x_{10}x_{31}+x_{20}x_{01} \\
&+4x_{20}x_{11}+x_{20}x_{31}+3x_{30}x_{01}+x_{30}x_{11}+x_{30}x_{21}+2x_{01}x_{12}+x_{01}x_{22} \\
&+3x_{01}x_{32}+2x_{11}x_{02}+4x_{11}x_{22}+x_{11}x_{32}+x_{21}x_{02}+4x_{21}x_{12}+x_{21}x_{32} \\
&+3x_{31}x_{02}+x_{31}x_{12}+x_{31}x_{22}+2x_{02}x_{13}+x_{02}x_{23}+3x_{02}x_{33}+2x_{12}x_{03} \\
&+4x_{12}x_{23}+x_{12}x_{33}+x_{22}x_{03}+4x_{22}x_{13}+x_{22}x_{33}+3x_{32}x_{03}+x_{32}x_{13}+x_{32}x_{23}.
\end{aligned}$$

Chapter 4, Adiabatic Quantum Computing and Quantum Annealing

(4.1) We first consider a state $|x\rangle = |x_0\rangle|x_1\rangle\cdots|x_{n-1}\rangle$ where each $|x_j\rangle$ is either $|+\rangle$ or $|-\rangle$. The set of all 2^n such states for an orthonormal basis and, hence, any generic state $|\psi\rangle$ can be written as $|\psi\rangle = \sum_x a_x |x\rangle$, where $\sum |a_x|^2 = 1$.

Then, for each j, it holds that

$$\langle x| X_j |x\rangle = \langle x_j| X_j |x_j\rangle .$$

But $\langle x_j| X_j |x_j\rangle$ is 1 if $|x_j\rangle = |+\rangle$ and it is -1 if $|x_j\rangle = |-\rangle$. Hence, it holds that

$$\langle\psi| X_j |\psi\rangle = \sum_x |a_x|^2 \langle x| X_j |x\rangle \leq 1,$$

because $|a_x|^2 \geq 0$ for every x and $\sum |a_x|^2 = 1$.

Then, since $H_0 = -\sum_{j=0}^{n-1} X_j$, by linearity we have

$$\langle\psi| H_0 |\psi\rangle = -\sum_{j=0}^{n-1} \langle\psi| X_j |\psi\rangle \geq -n.$$

On the other hand, if we consider $|\psi_0\rangle = \bigotimes_{i=0}^{n-1} |+\rangle$, by the previous reasoning we have that $\langle\psi_0| X_j |\psi_0\rangle = 1$. Hence, $\langle\psi_0| H_0 |\psi_0\rangle = -n$, which is the minimum possible value and, hence, $|\psi_0\rangle$ is the ground state that we were looking for.

(4.2) We can define the QUBO problem of minimizing $x_0x_2 - x_0x_1 + 2x_1$ with the following code:

```python
import dimod
J = {(0,1):-1, (0,2):1}
h = {1:2}
problem = dimod.BinaryQuadraticModel(h, J, 0.0, dimod.BINARY)
print("The problem we are going to solve is:")
print(problem)
```

We can then solve it with the following:

```python
from dwave.system import DWaveSampler
from dwave.system import EmbeddingComposite
sampler = EmbeddingComposite(DWaveSampler())
result = sampler.sample(problem, num_reads=10)
print("The solutions that we have obtained are")
print(result)
```

(4.3) For simplicity, we will denote the slack variables as s_0 and s_1. Then, the penalty term is $(y_0 + 2y_1 + s_0 + s_1 - 2)^2$. When you multiply by 5, expand it, and add it to the cost function, you obtain exactly the expression computed by the `cqm_to_bqm` method.

(4.4) For qubits 0 through 7, we have the following connections:

```
{0: {4, 5, 6, 7, 128}, 1: {4, 5, 6, 7, 129},
 2: {4, 5, 6, 7, 130}, 3: {4, 5, 6, 7, 131},
 4: {0, 1, 2, 3, 12}, 5: {0, 1, 2, 3, 13},
 6: {0, 1, 2, 3, 14}, 7: {0, 1, 2, 3, 15}}
```

Clearly, each vertex from 0 to 3 is connected to each from 4 to 7, as we need. Moreover, each vertex from 0 to 3 is connected to one vertex from 128 to 131, which are in the cell below the first one, and each vertex from 4 to 7 is connected to one vertex from 12 to 15, which are in the cell to the right of the first one.

(4.5) You can easily check those values with the following instructions:

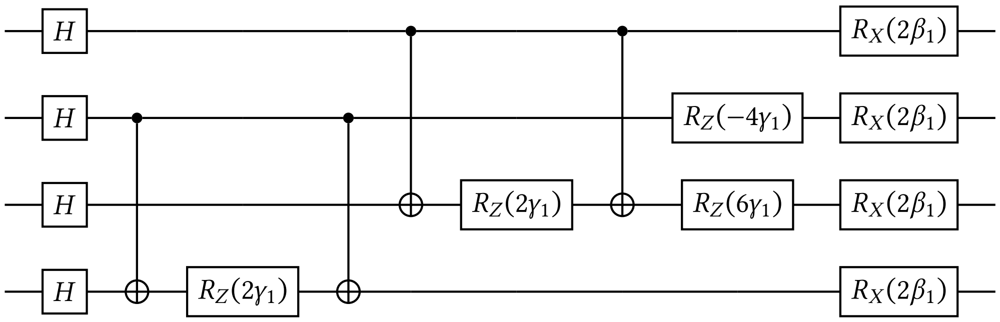
```
sampler = DWaveSampler(solver = "DW_2000Q_6")
print("The default annealing time is",
    sampler.properties["default_annealing_time"],"microsends")
print("The possible values for the annealing time (in microseconds)"\
    " lie in the range",sampler.properties["annealing_time_range"])
```

The output, in this case, will be as follows:

```
The default annealing time is 20.0 microsends
The possible values for the annealing time (in microseconds)
    lie in the range [1.0, 2000.0]
```

Chapter 5, QAOA: Quantum Approximate Optimization Algorithm

(5.1) The QAOA circuit for $Z_1Z_3 + Z_0Z_2 - 2Z_1 + 3Z_2$ with $p = 1$ is the following:



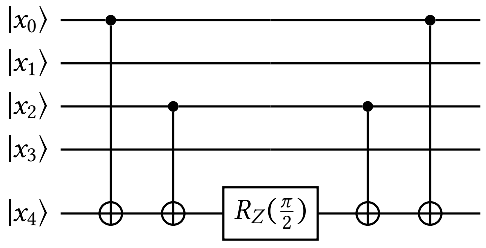
(5.2) It holds that

$$\langle 100| H_1 |100\rangle = 3\langle 100| Z_0Z_2 |100\rangle - \langle 100| Z_1Z_2 |100\rangle + 2\langle 100| Z_0 |100\rangle = -3 - 1 - 2 = -6.$$

(5.3) We can rewrite the problem as

$$\begin{array}{ll} \text{Minimize} & (1-x_0)(1-x_1)x_2(1-x_3) + x_0(1-x_1)(1-x_2)(1-x_3) + x_0(1-x_1)x_2x_3 \\ \text{subject to} & x_j \in \{0,1\}, \qquad j = 0,1,2,3. \end{array}$$

(5.4) The operation can be implemented with the following circuit:



(5.5) It holds that

$$\langle 100| H_1 |100\rangle = \langle 100| Z_0Z_1Z_2 |100\rangle + 3\langle 100| Z_0Z_2 |100\rangle - \langle 100| Z_1Z_2 |100\rangle$$
$$+ 2\langle 100| Z_0 |100\rangle = -1 - 3 - 1 - 2 = -7.$$

(5.6) You can use the following code to obtain reproducible results:

```
from qiskit import Aer
from qiskit.algorithms import QAOA
from qiskit.algorithms.optimizers import COBYLA
from qiskit.utils import algorithm_globals, QuantumInstance
from qiskit_optimization.algorithms import MinimumEigenOptimizer

seed = 1234
algorithm_globals.random_seed = seed
quantum_instance = QuantumInstance(Aer.get_backend("aer_simulator"),
    shots = 1024, seed_simulator=seed, seed_transpiler=seed)
qaoa = QAOA(optimizer = COBYLA(),
            quantum_instance=quantum_instance, reps = 1)
qaoa_optimizer = MinimumEigenOptimizer(qaoa)
result = qaoa_optimizer.solve(qp)
print('Variable order:', [var.name for var in result.variables])
for s in result.samples:
    print(s)
```

(5.7) We can define the $-3Z_0Z_1Z_2 + 2Z_1Z_2 - Z_2$ Hamiltonian using the following instructions:

```
coefficients = [-3,2,-1]
paulis = [PauliZ(0)@PauliZ(1)@PauliZ(2),
    PauliZ(1)@PauliZ(2),PauliZ(2)]
```

```
H = qml.Hamiltonian(coefficients,paulis)
```

We can also use

```
H = -3*PauliZ(0)@PauliZ(1)@PauliZ(2)
    + 2*PauliZ(1)@PauliZ(2) -PauliZ(2)
```

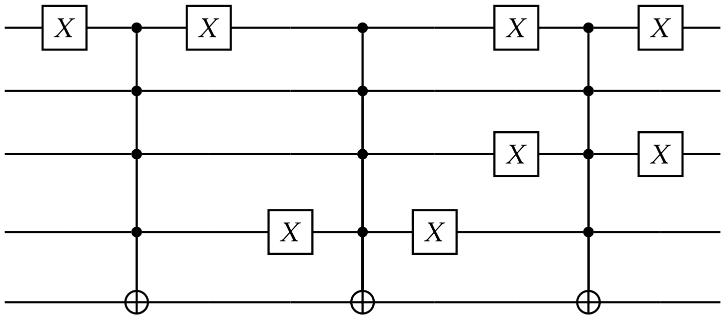
Chapter 6, GAS: Grover Adaptative Search

(6.1) From our definition, O_f always takes one basis state to another basis state. Hence, in matrix representation, its columns are vectors in which exactly one element is 1 and the rest are 0. This means that, in particular, all its entries are real.

What is more, this matrix is symmetric. To prove this, suppose that the matrix has entries m_{jk}. If it is not symmetric, there exist j, k such that $m_{jk} \neq m_{kj}$. We can suppose, without loss of generality, that $m_{jk} = 0$ and $m_{kj} = 1$. We also know that $O_f O_f = I$, so the square of the matrix is the identity. In particular, $\sum_l m_{jl} m_{lj} = 1$, because this is the element in row j, column j of the square of the matrix. But we know that $m_{jk} m_{kj} = 0 \cdot 1 = 0$ and that $m_{lj} = 0$ if $l \neq k$, because there is a single 1 in each column. Nevertheless, then, $\sum_l m_{jl} m_{lj} = 0$, which is a contradiction.

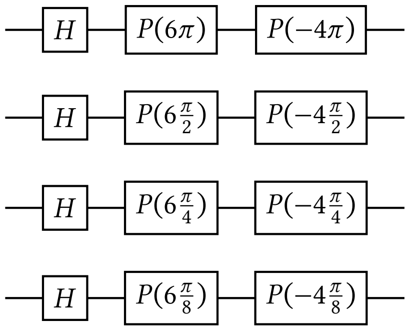
Thus, we have $O_f^\dagger = O_f$ and, since $O_f O_f = I$, it follows that O_f is unitary.

(6.2) We can use the following circuit:

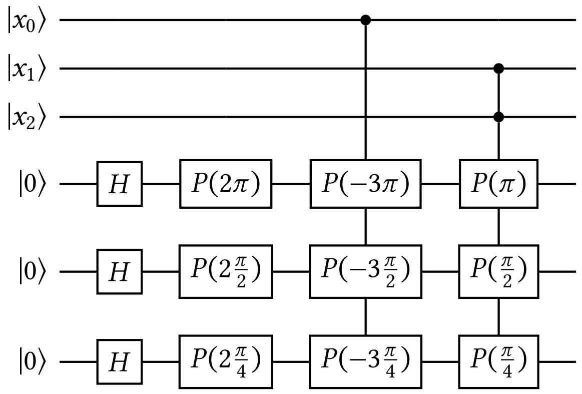


(6.3) The representation of 10 is 01010 and the representation of −7 is 11001. Their addition is 00011, which encodes 3.

(6.4) We can use the following circuit:



(6.5) We can use the following circuit:



(6.6) We can use the following code:

```python
from qiskit_optimization.problems import QuadraticProgram
from qiskit_optimization.algorithms import GroverOptimizer
from qiskit import Aer
from qiskit.utils import algorithm_globals, QuantumInstance
seed = 1234
algorithm_globals.random_seed = seed
```

```
qp = QuadraticProgram()
qp.binary_var('x')
qp.binary_var('y')
qp.binary_var('z')
qp.minimize(linear = {'x':3,'y':2,'z':-3}, quadratic = {('x','y'):3})

quantum_instance = QuantumInstance(Aer.get_backend("aer_simulator"),
    shots = 1024, seed_simulator = seed, seed_transpiler=seed)
grover_optimizer = GroverOptimizer(num_value_qubits = 5,
    num_iterations=4, quantum_instance=quantum_instance)
results = grover_optimizer.solve(qp)
print(results)
```

Chapter 7, VQE: Variational Quantum Eigensolver

(7.1) This all follows from the fact that the matrix is diagonal and all its diagonal entries are different, with eigenvalues corresponding to the actual labels of the measurement outcomes.

Remember that, if the coordinate matrix of an operator with respect to a basis is diagonal, this means that the basis vectors are eigenvectors of the operator and, what is more, the corresponding eigenvalues are found on the diagonal.

(7.2) We know that

$$(A_1 \otimes \cdots \otimes A_n) \left|\lambda_1\right\rangle \otimes \cdots \otimes \left|\lambda_n\right\rangle = A_1 \left|\lambda_1\right\rangle \otimes \cdots \otimes A_n \left|\lambda_n\right\rangle .$$

Since $A_j \left|\lambda_j\right\rangle = \lambda_j \left|\lambda_j\right\rangle$, the result follows directly.

(7.3) It holds that $Z\left|0\right\rangle = \left|0\right\rangle$, $Z\left|1\right\rangle = -\left|1\right\rangle$, $X\left|+\right\rangle = \left|+\right\rangle$, $X\left|-\right\rangle = -\left|-\right\rangle$, $Y\left(1/\sqrt{2}\right)(\left|0\right\rangle + i\left|1\right\rangle) = \left(1/\sqrt{2}\right)(\left|0\right\rangle + i\left|1\right\rangle)$, and $Y\left(1/\sqrt{2}\right)(\left|0\right\rangle - i\left|1\right\rangle) = -\left(1/\sqrt{2}\right)(\left|0\right\rangle - i\left|1\right\rangle)$.

Since it is also true that $I\left|\psi\right\rangle = \left|\psi\right\rangle$ for any $\left|\psi\right\rangle$, the result follows.

(7.4) A possible orthonormal basis of eigenvectors of $Z \otimes I \otimes X$ is formed by $\left|0\right\rangle\left|0\right\rangle\left|+\right\rangle$, $\left|0\right\rangle\left|1\right\rangle\left|-\right\rangle$, $\left|1\right\rangle\left|0\right\rangle\left|-\right\rangle$, $\left|1\right\rangle\left|1\right\rangle\left|+\right\rangle$, $\left|0\right\rangle\left|0\right\rangle\left|-\right\rangle$, $\left|0\right\rangle\left|1\right\rangle\left|+\right\rangle$, $\left|1\right\rangle\left|0\right\rangle\left|+\right\rangle$ and $\left|1\right\rangle\left|1\right\rangle\left|-\right\rangle$. The first four eigenvectors are associated with the 1 eigenvalue, and the rest with the -1 eigenvalue.

Let's denote, for simplicity, $\left|i\right\rangle = \left(1/\sqrt{2}\right)\left(\left|0\right\rangle + i\left|1\right\rangle\right)$ and $\left|-i\right\rangle = \left(1/\sqrt{2}\right)\left(\left|0\right\rangle - i\left|1\right\rangle\right)$. Then, a possible orthonormal basis of eigenvalues for $I \otimes Y \otimes Y$ is $\left|0\right\rangle\left|i\right\rangle\left|i\right\rangle$, $\left|0\right\rangle\left|-i\right\rangle\left|-i\right\rangle$, $\left|1\right\rangle\left|i\right\rangle\left|-i\right\rangle$, $\left|1\right\rangle\left|-i\right\rangle\left|i\right\rangle$, $\left|0\right\rangle\left|i\right\rangle\left|-i\right\rangle$, $\left|0\right\rangle\left|-i\right\rangle\left|i\right\rangle$, $\left|1\right\rangle\left|i\right\rangle\left|i\right\rangle$ and $\left|1\right\rangle\left|-i\right\rangle\left|-i\right\rangle$. The first four eigenvectors are associated to the 1 eigenvalue and the rest, to the -1 eigenvalue.

(7.5) It holds that $H\left|0\right\rangle = \left|+\right\rangle$ and $H\left|1\right\rangle = \left|-\right\rangle$. This proves that H takes the computational basis to the eigenvectors of X. Also, $SH\left|0\right\rangle = \left(1/\sqrt{2}\right)\left(\left|0\right\rangle + i\left|1\right\rangle\right)$ and $SH\left|1\right\rangle = \left(1/\sqrt{2}\right)\left(\left|0\right\rangle - i\left|1\right\rangle\right)$, so SH takes the computational basis to the eigenvectors of Y.

(7.6) This follows directly from the fact that if $\{\left|u_j\right\rangle\}_j$ and $\{\left|v_k\right\rangle\}_k$ are eigenvector bases of A_1 and A_2, respectively, then $\{\left|u_j\right\rangle \otimes \left|v_k\right\rangle\}_{j,k}$ is an eigenvector basis of $A_1 \otimes A_2$.

(7.7) The Hamiltonian for our problem is

$$H = Z_0Z_1 + Z_1Z_2 + Z_2Z_3 + Z_3Z_4 + Z_4Z_0.$$

Then, we can use the following code to solve it with VQE:

```
from qiskit.circuit.library import EfficientSU2
from qiskit.algorithms import VQE
from qiskit import Aer
from qiskit.utils import QuantumInstance
import numpy as np
from qiskit.algorithms.optimizers import COBYLA
from qiskit.opflow import Z, I


seed = 1234
```

```
np.random.seed(seed)

H= (Z^Z^I^I^I) + (I^Z^Z^I^I) + (I^I^Z^Z^I) + (I^I^I^Z^Z) + (Z^I^I^I^Z)

ansatz = EfficientSU2(num_qubits=5, reps=1, entanglement="linear",
    insert_barriers = True)
optimizer = COBYLA()
initial_point = np.random.random(ansatz.num_parameters)
quantum_instance = QuantumInstance(backend =
    Aer.get_backend('aer_simulator_statevector'))
vqe = VQE(ansatz=ansatz, optimizer=optimizer,
    initial_point=initial_point,
    quantum_instance=quantum_instance)
result = vqe.compute_minimum_eigenvalue(H)
print(result)
```

(7.8) We can use the following code:

```
from qiskit_nature.drivers import Molecule
from qiskit_nature.drivers.second_quantization import \
    ElectronicStructureMoleculeDriver, ElectronicStructureDriverType
from qiskit_nature.problems.second_quantization import \
    ElectronicStructureProblem
from qiskit_nature.converters.second_quantization import QubitConverter
from qiskit_nature.mappers.second_quantization import JordanWignerMapper
from qiskit_nature.algorithms import VQEUCCFactory
from qiskit import Aer
from qiskit.utils import QuantumInstance
from qiskit_nature.algorithms import GroundStateEigensolver
import matplotlib.pyplot as plt
import numpy as np

quantum_instance = QuantumInstance(
    backend = Aer.get_backend('aer_simulator_statevector'))

vqeuccf = VQEUCCFactory(quantum_instance = quantum_instance)

qconverter = QubitConverter(JordanWignerMapper())
solver = GroundStateEigensolver(qconverter, vqeuccf)

energies = []
distances = np.arange(0.2, 2.01, 0.01)
for d in distances:
  mol = Molecule(geometry=[['H', [0., 0., -d/2]],
                            ['H', [0., 0., d/2]]])

  driver = ElectronicStructureMoleculeDriver(mol, basis='sto3g',
        driver_type=ElectronicStructureDriverType.PYSCF)
  problem = ElectronicStructureProblem(driver)
  result = solver.solve(problem)
  energies.append(result.total_energies)

plt.plot(distances, energies)
plt.title('Dissociation profile')
plt.xlabel('Distance')
plt.ylabel('Energy');
```

(7.9) We can use the following code:

```
from qiskit import *
from qiskit.providers.aer import AerSimulator
from qiskit.utils.mitigation import CompleteMeasFitter
from qiskit.utils import QuantumInstance

provider = IBMQ.load_account()
backend = AerSimulator.from_backend(
    provider.get_backend('ibmq_manila'))

shots = 1024

qc = QuantumCircuit(2,2)
qc.h(0)
qc.cx(0,1)
qc.measure(range(2),range(2))

result = execute(qc, backend, shots = shots)
print("Result of noisy simulation:")
print(result.result().get_counts())

quantum_instance = QuantumInstance(
    backend = backend, shots = shots,
    measurement_error_mitigation_cls=CompleteMeasFitter)

result = quantum_instance.execute(qc)
print("Result of noisy simulation with error mitigation:")
print(result.get_counts())
```

Our results when running these instructions were the following:

```
Result of noisy simulation:
```

```
{'01': 88, '10': 50, '00': 453, '11': 433}
Result of noisy simulation with error mitigation:
{'00': 475, '01': 12, '10': 14, '11': 523}
```

We know that the ideal result of running this circuit should not produce any 01 or 10 measurement. These are present quite prominently in the noisy simulation, but not so much when we use readout error mitigation.

(7.10) We can use the following code:

```
from qiskit.opflow import Z
from qiskit.providers.aer import AerSimulator
from qiskit.algorithms import QAOA
from qiskit.utils import QuantumInstance
from qiskit import Aer, IBMQ
from qiskit.algorithms.optimizers import COBYLA
from qiskit.utils.mitigation import CompleteMeasFitter

H1 = Z^Z

provider = IBMQ.load_account()
backend = AerSimulator.from_backend(
    provider.get_backend('ibmq_manila'))

quantum_instance = QuantumInstance(backend=backend,
                   shots = 1024)
qaoa = QAOA(optimizer = COBYLA(), quantum_instance=quantum_instance)
result = qaoa.compute_minimum_eigenvalue(H1)
print("Result of noisy simulation:",result.optimal_value)

quantum_instance = QuantumInstance(backend=backend,
    measurement_error_mitigation_cls=CompleteMeasFitter,
    shots = 1024)
qaoa = QAOA(optimizer = COBYLA(), quantum_instance=quantum_instance)
result = qaoa.compute_minimum_eigenvalue(H1)
print("Result of noisy simulation with error mitigation:",
    result.optimal_value)
```

The results that we obtained when we ran it were the following:

```
Result of noisy simulation: -0.8066406250000001
Result of noisy simulation with error mitigation: -0.93359375
```

We know that the actual optimal value for our Hamiltonian is -1. We observe, then, that noise has a negative effect on the performance of QAOA in this case and that it is reduced by the use of readout error mitigation.

Chapter 8, What is Quantum machine Learning?

(8.1) Let's proceed by a proof by contradiction. We will assume that there exist some coefficients w_1, w_2, b such that

$$0w_1 + 1w_2 + b = 1, \qquad 1w_1 + 0w_2 + b = 1,$$

$$0w_1 + 0w_2 + b = 0, \qquad 1w_1 + 1w_2 + b = 0.$$

Simplifying, these are equivalent to

$$w_2 + b = 1, \qquad w_1 + b = 1, \qquad b = 0, \qquad w_1 + w_2 + b = 0.$$

The first three identities imply that $b = 0$ and $w_1 = w_2 = 1$, hence the last identity cannot be satisfied.

(8.2) Histograms are usually versatile and powerful options. In this case, however, since our dataset has two features, we could have also drawn a scatter plot using the `plt.scatter` function.

(8.3) The function is clearly strictly increasing, for its derivative is

$$\frac{e^x}{(e^x+1)^2} > 0.$$

Moreover, it is immediate that $\lim_{x\to\infty} S(x) = 1$ and $\lim_{x\to-\infty} S(x) = 0$.

The ELU function is smooth because the derivative of x at 0 is 1 and so is that of $e^x - 1$ at 0. Both functions are strictly increasing and $\lim_{x\to\infty} x = \infty$ and $\lim_{x\to-\infty} e^x - 1 = -1$.

The image of the ReLU function is, clearly, $[0, \infty)$. It is not smooth because $(x)' = 1$ yet $(0)' = 0$.

(8.4) Without loss of generality, we will assume $y = 1$ (the case $y = 0$ is fully analogous). If $M_\theta(x) = y = 1$, then $H(\theta; x, y) = -1\log(1) + 0 = 0$ because $1 - y$ is 0 for any value of x. On the other hand, when $M_\theta(x) \to 0$, then $-\log(M_\theta(x)) \to \infty$, hence H also diverges.

(8.5) We can plot the losses using the following piece of code:

```
val_loss = history.history["val_loss"]
train_loss = history.history["loss"]
epochs = range(len(train_loss))
plt.plot(epochs, train_loss, label = "Training loss")
plt.plot(epochs, val_loss, label = "Validation loss")
plt.legend()
plt.show()
```

(8.6) The result is less accurate when we decrease the learning rate without increasing the number of epochs, because the algorithm can't take enough steps to reach a minimum. We also get worse results when we reduce the training dataset to 20, because we have overfitting. This can be identified by looking at the evolution of the validation loss and noticing how it skyrockets while the training loss plummets.

Chapter 9, Quantum Support Vector Machines

(9.1) We will prove that the distance between a hyperplane H_1, characterized by either $\vec{w}\cdot\vec{x}+b=1$ or $\vec{w}\cdot\vec{x}+b=-1$, and H_0, given by $\vec{w}\cdot\vec{x}+b=0$, is $1/\|w\|$. The result will then follow from the fact that $\vec{w}\cdot\vec{x}+b=\pm1$ are projections of each other over H_0.

Let us consider a point $\vec{x}_0 \in H_0$. The distance between H_0 and H_1 will be the length of the only vector in the direction normal to H_0 that connects $\vec{x}_0$ to a point in H_1. What is more, since $\vec{w}$ is normal to H_0, such a vector needs to be of the form $\alpha\vec{w}$ for some scalar α. Let us find that scalar.

We know that $\vec{x}_0+\alpha\vec{x}_1 \in H_1$, so we must have

$$\vec{w}\cdot\left(\vec{x}_0+\alpha\vec{w}\right)+b=1.$$

But taking into account that $x_0 \in H_0$ and, therefore, $\vec{w}\cdot\vec{x}_0+b=0$, this can be further simplified to

$$\vec{w}\cdot\alpha\vec{w}=1 \iff \alpha=\frac{1}{\|w\|^2}.$$

The length of $\alpha\vec{w}$ will just be $|\alpha|\cdot\left\|\vec{w}\right\|$, which is $1/\|w\|$, just as we wanted to prove.

(9.2) Let's assume that the kernel function is defined as $k(a,b)=|\langle\varphi(a)|\varphi(b)\rangle|^2$. The inner product in $\mathbb{C}^n$ is conjugate-symmetric, so we must have

$$k(b,a)=|\langle\varphi(b)|\varphi(a)\rangle|^2=\left|\overline{\langle\varphi(a)|\varphi(b)\rangle}\right|^2=|\langle\varphi(a)|\varphi(b)\rangle|^2=k(a,b).$$

(9.3) Quantum states need to be normalized and, therefore, the scalar product with themselves must be one.

(9.4) The following piece of code would implement that function:

```
from qiskit import *
from qiskit . circuit import ParameterVector

def AngleEncodingX(n):
    x = ParameterVector("x", length = n)
    qc = QuantumCircuit(n)
    for i in range(n):
        qc.rx(parameter[i], i)
    return qc
```

Chapter 10, Quantum Neural Networks

(10.1) It suffices to remember that, while kets are represented by column matrices, bras are represented by row matrices. In this way,

$$|0\rangle\langle 0| = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$

Analogously,

$$|1\rangle\langle 1| = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

The results follow trivially from this.

(10.2) The process would be fully analogous. The only differences would be found in the definition of the network, the device, and the weights dictionary, which could be the following:

```
nqubits = 5
dev = qml.device("default.qubit", wires=nqubits)

def qnn_circuit(inputs, theta):
    qml.AmplitudeEmbedding(features = [a for a in inputs],
        wires = range(nqubits), normalize = True, pad_with = 0.)
    TwoLocal(nqubits = nqubits, theta = theta, reps = 2)
    return qml.expval(qml.Hermitian(M, wires = [0]))

qnn = qml.QNode(qnn_circuit, dev, interface="tf")

weights = {"theta": 15}
```

Also, remember that you should train this model on the original dataset (`x_tr` and `y_tr`), not on the reduced one!

(10.3) On quantum hardware, one may use — for most return types — the finite differences method and the parameter shift rule. On simulators — under certain conditions — one may use these methods in addition to backpropagation and adjoint differentiation.

Chapter 11, The Best of Both Worlds: Hybrid Architectures

(11.1) In order to include the extra classical layers, we would have to execute the same code but define the model in the following manner:

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(20),
    tf.keras.layers.Dense(16, activation = "elu"),
    tf.keras.layers.Dense(8, activation = "elu"),
    tf.keras.layers.Dense(4, activation = "sigmoid"),
    qml.qnn.KerasLayer(qnn, weights, output_dim=1)
])
```

This model has similar performance after training. The addition of the classical layers doesn't make a very significant difference.

(11.2) In order to have optimized both the learning rate and the batch size, we could have defined the objective function as follows:

```python
def objective(trial):
    # Define the learning rate as an optimizable parameter.
    lrate = trial.suggest_float("learning_rate", 0.001, 0.1)
    bsize = trial.suggest_int("batch_size", 5, 50)

    # Define the optimizer with the learning rate.
    opt = tf.keras.optimizers.Adam(learning_rate = lrate)

    # Prepare and compile the model.
    model = tf.keras.models.Sequential([
        tf.keras.layers.Input(20),
        tf.keras.layers.Dense(4, activation = "sigmoid"),
        qml.qnn.KerasLayer(qnn, weights, output_dim=1)
    ])
    model.compile(opt, loss=tf.keras.losses.BinaryCrossentropy())

    # Train it!
    history = model.fit(x_tr, y_tr, epochs = 50, shuffle = True,
        validation_data = (x_val, y_val),
        batch_size = bsize,
        callbacks = [earlystop],
        verbose = 0 # We want TensorFlow to be quiet.
    )

    # Return the validation accuracy.
    return accuracy_score(model.predict(x_val) >= 0.5, y_val)
```

(11.3) We could try to find a numerical approximation of $f(x) = (x-3)^2$ using Optuna as follows:

```
import optuna
from optuna.samplers import TPESampler

seed = 1234

def objective(trial):
    x = trial.suggest_float("x", -10, 10)
    return (x-3)**2

study = optuna.create_study(direction='minimize',
sampler=TPESampler(seed = seed))
study.optimize(objective, n_trials=100)
```

Of course, Optuna was not designed with (general) function minimization in mind, but it was conceived solely as a hyperparameter optimizer.

(11.4) Let $y = 0, 1$ be the expected label. It suffices to notice that y in one-hot form is $(1-y, y)$ and that the probability assigned by the model to x being of class y is $N_\theta(x)_j$. Hence

$$\begin{aligned} H(\theta; x, (1-y, y)) &= \sum_j -y_j \log\big(N_\theta(x)_j\big) \\ &= -(1-y)\log(N_\theta(x)_0) - y\log(N_\theta(x)_1) \\ &= -(1-y)\log(1-N_\theta(x)_1) - y\log(N_\theta(x)_1), \end{aligned}$$

where we assume that $N_\theta(x)$ is normalized and, therefore, $N_\theta(x)_0 + N_\theta(x)_1 = 1$. The result now follows from the fact that, in binary cross entropy, the probability that we consider is that of assigning label 1, that is, $N_(\theta)_1$.

(11.5) We just have to use the y targets instead of the y_hot targets when preparing the datasets and then call the sparse categorical cross-entropy loss given in the statement when compiling the model.

(11.6) You can create a suitable dataset with 1000 samples and 20 features with the following instruction:

```
x, y = make_regression(n_samples = 1000, n_features = 20)
```

Then, you can construct the model as follows:

```
nqubits = 4
dev = qml.device("lightning.qubit", wires = nqubits)

@qml.qnode(dev, interface="tf", diff_method = "adjoint")
def qnn(inputs, theta):
    qml.AngleEmbedding(inputs, range(nqubits))
    TwoLocal(nqubits, theta, reps = 2)
    return [qml.expval(qml.Hermitian(M, wires = [0]))]

weights = {"theta": 12}

model = tf.keras.models.Sequential([
    tf.keras.layers.Input(20),
    tf.keras.layers.Dense(16, activation = "elu"),
    tf.keras.layers.Dense(8, activation = "elu"),
    tf.keras.layers.Dense(4, activation = "sigmoid"),
    qml.qnn.KerasLayer(qnn, weights, output_dim=1),
    tf.keras.layers.Dense(1)
])
```

Then it would be trained like any of our previous models using the MSE loss function, which you can access with `tf.keras.losses.MeanSquaredError()`.

Chapter 12, Quantum Generative Adversarial Networks

(12.1) (1) QSVMs, QNNs or Hybrid QNNs (2) QGANs. (3) QSVMs, QNNs or Hybrid QNNs. (4) QSVMs, QNNs or Hybrid QNNs. (5) QGANs.

(12.2) The steps to produce the solution are analogous to what we did in the main text; it's only necessary to change the values of the angles that define the state $|\psi_1\rangle$, and, possibly, to increase the number of training cycles.

(12.3) Let's consider a point (x, y, z) in the Bloch sphere. Its spherical coordinates are (θ, φ) such that

$$(x, y, z) = (\sin\theta\cos\varphi, \sin\theta\sin\varphi, \cos\theta)$$

and a state with those spherical Bloch sphere coordinates is in the state

$$|\psi\rangle = \cos(\theta/2)\,|0\rangle + e^{i\varphi}\sin(\theta/2)\,|1\rangle\,.$$

The expectation value of $Z = |0\rangle\langle 0| - |1\rangle\langle 1|$ in the state $|\psi\rangle$ is

$$\langle\psi|\,Z\,|\psi\rangle = \cos^2(\theta/2) - e^{-i\varphi+i\varphi}\sin^2(\theta/2) = \cos\theta = z.$$

Regarding the expectation value $Y = i\,|1\rangle\langle 0| - i\,|0\rangle\langle 1|$, we have

$$\begin{aligned}\langle\psi|\,Y\,|\psi\rangle &= ie^{-i\varphi}\sin(\theta/2)\cos(\theta/2) - ie^{i\varphi}\sin(\theta/2)\cos(\theta/2)\\ &= i(e^{-i\varphi} - e^{i\varphi})(\sin(\theta/2)\cos(\theta/2))\\ &= \sin\varphi\cdot 2\sin(\theta/2)\cos(\theta/2) = \sin\varphi\sin\theta = y.\end{aligned}$$

Lastly, in regard to the expectation value of $X = |1\rangle\langle 0| + |0\rangle\langle 1|$,

$$\begin{aligned}\langle\psi| X |\psi\rangle &= e^{-i\varphi}\sin(\theta/2)\cos(\theta/2) + e^{i\varphi}\sin(\theta/2)\cos(\theta/2)\\ &= (e^{-i\varphi} + e^{i\varphi})(\sin(\theta/2)\cos(\theta/2))\\ &= \cos\varphi \cdot 2\sin(\theta/2)\cos(\theta/2) = \cos\varphi\sin\theta = x.\end{aligned}$$