Session 3: Execution of real quantum circuits on IBM's computers

[2]:
```
# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2, EstimatorV2
from qiskit.quantum_info import SparsePauliOp, Operator

# Other imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In the lab session we used premium accounts to fast forward over the long queue list of IBM's quantum machines. However at home we can use our free accounts to schedule calculations.

[3]:
```
# Recover previous session information
df = pd.read_csv('session.csv')

# Loading account
service = QiskitRuntimeService(channel='ibm_quantum', token=df.token[0])

# Simulation machine
backend_S = AerSimulator()
# In the lab we used backend_S = service.backend('ibmq_qasm_simulator'),
# however cloud simulators have been deprecated and will be removed on 15 May␣
↪2024
```

[ ]:
```
# Quantum machine
backend_Q = service.least_busy(operational=True, simulator=False,␣
↪min_num_qubits=20)

# Save backend
df.backend_Q = backend_Q.name
df.to_csv('session.csv', index=False)
```

[3]:
```
backend_Q = service.get_backend(df.backend_Q[0])
backend_Q.status()
```

[3]:
```
<qiskit.providers.models.backendstatus.BackendStatus at 0x2109434b090>
```

[4]:
```
# Maximum execution time in seconds
my_options = {'max_execution_time': 30}

# Classical and quantum estimators
estimator_S = EstimatorV2(backend=backend_S)
```

```
estimator_Q = EstimatorV2(backend=backend_Q, options=my_options)

# Classical and quantum samplers
sampler_S = SamplerV2(backend=backend_S)
sampler_Q = SamplerV2(backend=backend_Q, options=my_options)
```

Random numbers

The generation of random numbers using a classical simulator of a quantum machine was conducted during the second session. Now, let's generate "true random numbers"! To achieve this, we define a 3-qubit circuit that prepares each of them in the superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and measures them in the computational basis $|0\rangle, |1\rangle$. This setup ensures that the measurement outcomes will be 0 or 1 with equal probabilities of 0.5.

Question 1: Study the definition of the circuit with care.

Answer: It creates a barrier of Hadamard gates, which prepares each qubit in an equal superposition of $|0\rangle$ and $|1\rangle$. The circuit assigns three qubits to each random number to be generated, meaning the random numbers range between 0 and 7. This is an example of the *Hadamard transform*, which in this case performs the following operation:

$$H^{\otimes 90}|0\rangle^{\otimes 90} = |+\rangle^{\otimes 90}$$

More generally, if we have a register initialised to a state $|j\rangle$ of the computational basis, the action of a Hadamard transform is the following:

$$H^{\otimes n}|j\rangle = \bigotimes_{i=1}^{n} H|j_i\rangle = \frac{1}{2^{n/2}} \bigotimes_{i=1}^{n}(|0\rangle + (-1)^{j_i}|1\rangle) = \frac{1}{2^{n/2}} \bigotimes_{i=1}^{n} \sum_{k_i=0}^{1} (-1)^{k_i j_i} |k_i\rangle =$$

$$= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} (-1)^{k_1 j_1} |k_1\rangle \sum_{k_2=0}^{1} (-1)^{k_2 j_2} |k_2\rangle ... \sum_{k_n=0}^{1} (-1)^{k_n j_n} |k_n\rangle =$$

$$= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} ... \sum_{k_n=0}^{1} (-1)^{\sum_{i=1}^{n} k_i j_i} |k_1 k_2 ... k_n\rangle \equiv$$

$$\equiv \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} (-1)^{\sum_{i=1}^{n} k_i j_i} |k\rangle$$

[100]:
```
# How many random numbers will be produced in a single shot:
Nnumbers = 30  # Must be smaller than [127 qubits/3]=42
# Prepare the input circuit
QRNG = QuantumCircuit(3*Nnumbers) # Three qubits per number
QRNG.h(range(3*Nnumbers)) # Apply Hadamard gate to each of the qubits
QRNG.measure_all() # Measure all qubits
```

```
QRNG = transpile(QRNG, backend=backend_Q) # Adapt circuit to architecture of␣
↪quantum machine
```

[34]:
```
# Execute the circuit directly on a quantum computer
job_qrng = sampler_Q.run([QRNG], shots=1)

# Save job identification code for later
df.qrng_id = job_qrng.job_id()
df.to_csv('session.csv', index=False)
```

```
c:\Users\zapat\Escritorio\CODE\PIE_Compu_Cuantica\venv\Lib\site-
packages\qiskit_ibm_runtime\qiskit_runtime_service.py:879: UserWarning: Your
current pending jobs are estimated to consume 649.5556236231932 quantum seconds,
but you only have 600 quantum seconds left in your monthly quota; therefore, it
is likely this job will be canceled
  warnings.warn(warning_message)
```

[81]:
```
# The following cell retrieves information about the job
job_qrng = service.job(df.qrng_id[0])
job_qrng.status()
```

[81]:
```
<JobStatus.DONE: 'job has successfully run'>
```

[82]:
```
# The next statement can put your session on hold until the job runs and returns
result_qrng = job_qrng.result()
```

Question 2: Comment on your results.

Answer: We obtained the following result from the IBM Quantum computer run in the lab with UCM tokens:

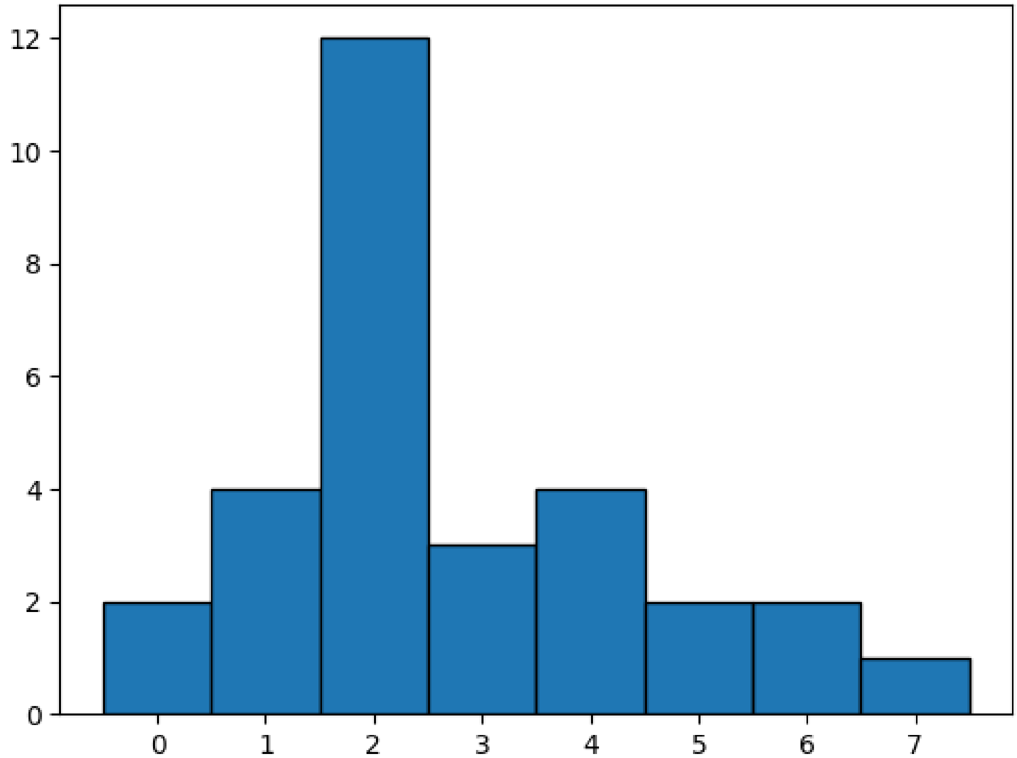
```
[{2367324008723302272311220054: 1.0}]
```

After depuration we obtain the following list of numbers

```
[6, 0, 2, 1, 6, 4, 4, 5, 2, 2, 3, 3, 4, 2, 2, 3, 2, 1, 2, 5, 2, 2, 2, 0, 2, 2, 7,
1, 4, 1]
```

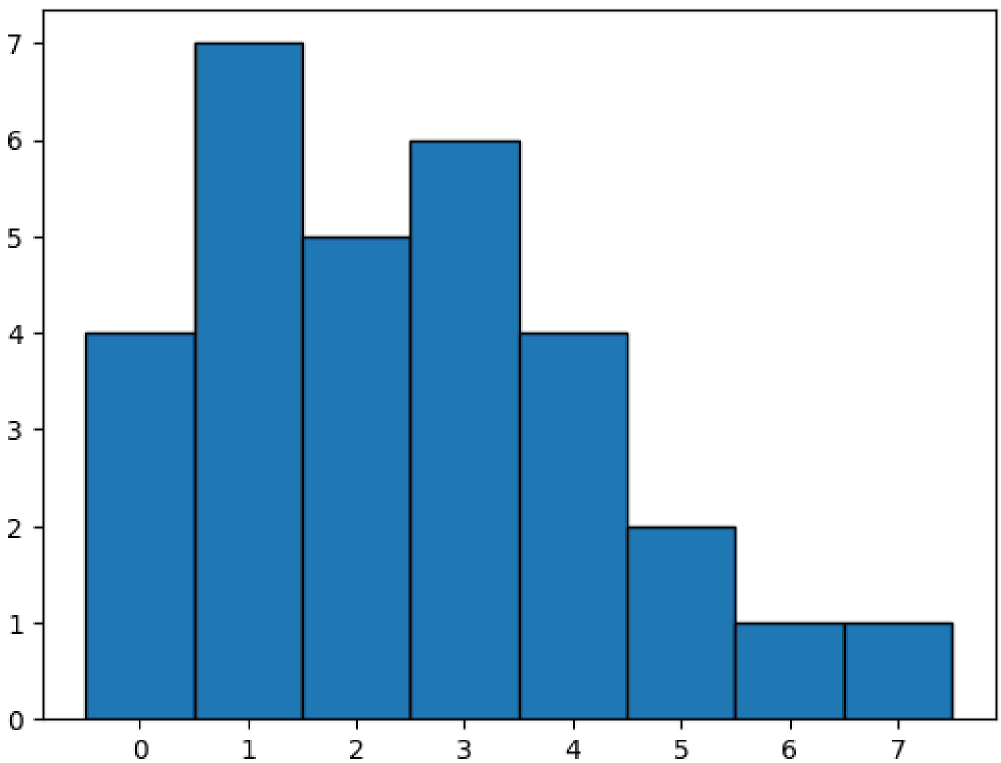
that we can plot in a histogram.

[7]:
```
RandomNumbers = [6, 0, 2, 1, 6, 4, 4, 5, 2, 2, 3, 3, 4, 2, 2, 3, 2, 1, 2, 5, 2,␣
↪2, 2, 0, 2, 2, 7, 1, 4, 1]
plt.hist(RandomNumbers, bins=range(9), align='left', edgecolor='black',␣
↪linewidth=1)
plt.show()
```



We can likewise retrieve our own results run at home.

[134]:
```
bits, = result_qrng[0].data.meas.get_counts()
numbers = [bits[i:i+3] for i in range(0, 3*Nnumbers, 3)]
decimal_numbers = [int(n, 2) for n in numbers]
plt.hist(decimal_numbers, bins=range(9), align='left', edgecolor='black',␣
 ↪linewidth=1)
plt.show()
```



The distribution is not “particularly uniform”, it rather looks more log-normal. However, we generated very few random numbers, to draw any statistical conclusions we would likely have to generate many more.

Deutsch’s algorithm

To encode the action of a Boolean fuction, Deutsch’s algorithm uses the following operation over two qubits

$$|x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle$$

where $\oplus$ denotes the binary addition (i.e. the addition mod. 2).

Homework 1 (to do later after the lab on pen and paper): Prove that this is a unitary operation.

Answer: It is clear that the quantum oracle preserves the norm, therefore it is unitary. If we take $(x, y) \in \{0, 1\}^2, f(x) \in \{0, 1\}$, we have that $y \oplus f(x) \in \{0, 1\}$ — in other words, $|x\rangle|y \oplus f(x)\rangle$ is a ket from the computational basis of the Hilbert space. The operation might be non-invertible if it mapped two different kets to a single one, but this is impossible, since $0 \oplus f(x) \neq 1 \oplus f(x)$:

$$(\langle x|\langle 0|)U_f^\dagger U_f(|x\rangle|1\rangle) = (\langle x|\langle 0 \oplus f(x)|)(|x\rangle|1 \oplus f(x)\rangle) = \langle x|x\rangle\langle 0 \oplus f(x)|1 \oplus f(x)\rangle = 0$$

Alternatively, given there exist a finite number of Boolean functions, we can calculate the oracle matrix for each f to better understand what U_f does. If we place the control qubit $|x\rangle$ first, as we'll later do in the circuits (so $|y,x\rangle \to |y \oplus f(x), x\rangle$), the representations of U_f are

$$f(x) = 0: \quad U_f = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad f(x) = 1: \quad U_f = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$f(x) = x: \quad U_f = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \quad f(x) = \neg x: \quad U_f = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

These matrices all represent permutations, which are unitary operations.

Question 2: Complete the following cells to create quantum circuits for Oracle_f that decides whether f is (or not) constant using the classical method with two evaluations.

Answer: To evaluate f using the oracle we can measure the leftmost qubit in the following two states:

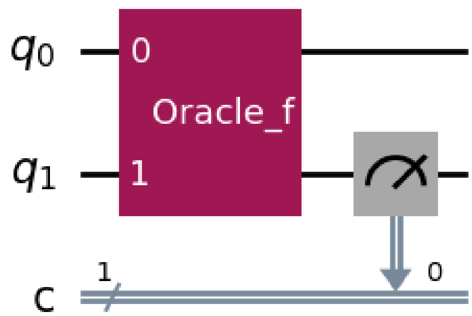
$$U_f|0\rangle|0\rangle = |f(0)\rangle|0\rangle$$

$$U_f|0\rangle|1\rangle = |f(1)\rangle|1\rangle$$

N.B.: Qubit significance and indexing in Qiskit goes from right to left, top to bottom, 0 to $n-1$.

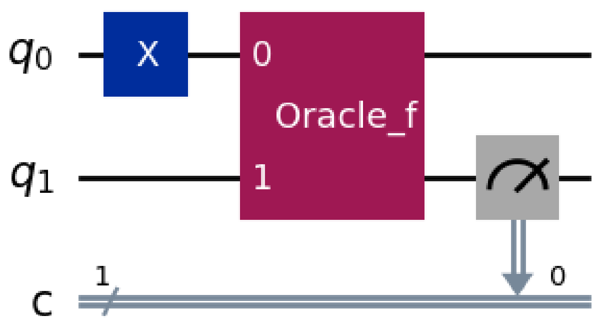
[9]:
```
Oracle_f = Operator([[0,0,1,0],[0,1,0,0],[1,0,0,0],[0,0,0,1]])
Oracle_g = Operator([[0,0,1,0],[0,0,0,1],[1,0,0,0],[0,1,0,0]])

# First evaluation
Check_f0 = QuantumCircuit(2,1)
Check_f0.unitary(Oracle_f, [0, 1], label='Oracle_f')
Check_f0.measure(1, 0)
Check_f0.draw(output='mpl')
```

[9]:



[123]:
```
# Second evaluation
Check_f1 = QuantumCircuit(2,1)
Check_f1.x(0)
Check_f1.unitary(Oracle_f, [0, 1], label='Oracle_f')
Check_f1.measure(1,0)
Check_f1.draw(output='mpl')
```

[123]:



[183]:
```
# Run both evaluations
job_clf = sampler_S.run([Check_f0, Check_f1], shots=1)
result_clf = job_clf.result()
```

Question 3: By printing "result_clf.quasi_dists", explain why the following cell gives the calculated result for $f(0)$ and $f(1)$.

Answer: The function $f(x)$ appears to be $\neg x$, and therefore a balanced function. Indeed, if we compare the `Oracle_f` with the matrix we calculated above for $f(x) = \neg x$, they coincide.

N.B.: For my calculations at home I used the `SamplerV2` class which has slightly different methods for retrieving results as compared to `Sampler`.

[184]:
```
# Print the measurement results
print(f'f(0) = {int(*result_clf[0].data.c.get_counts())}')
print(f'f(1) = {int(*result_clf[1].data.c.get_counts())}')
```

```
f(0) = 1
f(1) = 0
```

Question 4: Complete the following cells to create quantum circuits for Oracle_g that solve the problem of whether g is constant using the classical method of two evaluations.

Answer: We may repeat the previous process. We find that $g(x) = 1$, therefore g is a constant function.

[186]:
```
# First evaluation
Check_g0 = QuantumCircuit(2,1)
Check_g0.unitary(Oracle_g, [0, 1], label='Oracle_g')
Check_g0.measure(1, 0)

# Second evaluation
Check_g1 = QuantumCircuit(2,1)
Check_g1.x(0)
Check_g1.unitary(Oracle_g, [0, 1], label='Oracle_g')
Check_g1.measure(1,0)

# Run both evaluations
job_clg = sampler_S.run([Check_g0, Check_g1], shots=1)
result_clg = job_clg.result()

# Print the measurement results
print(f'g(0) = {int(*result_clg[0].data.c.get_counts())}')
print(f'g(1) = {int(*result_clg[1].data.c.get_counts())}')
```

```
g(0) = 1
g(1) = 1
```

The idea of Deutsch's algorithm is based on the use of quantum superposition to attempt a simultaneous evaluation and comparison of $f(0)$ and $f(1)$ with a single action of the quantum oracle.

If $f(0) = f(1)$,

$$U_f|+\rangle|-\rangle = \frac{1}{2}[|0\rangle|f(0)\rangle - |0\rangle|1 \oplus f(0)\rangle + |1\rangle|f(0)\rangle - |1\rangle|1 \oplus f(0)\rangle]$$

$$= \frac{1}{2}[(|0\rangle + |1\rangle)|f(0)\rangle - (|0\rangle + |1\rangle)|1 \oplus f(0)\rangle] = |+\rangle \left[\frac{|f(0)\rangle - |1 \oplus f(0)\rangle}{\sqrt{2}}\right]$$

Homework question 5: Prove (after the lab, on pen and paper): the alternative case, that if $f(0) \neq f(1)$, the final state becomes:

$$|-\rangle \left[\frac{|f(0)\rangle - |f(1)\rangle}{\sqrt{2}}\right].$$

Answer: If $f(0) \neq f(1)$, then $f(0) \oplus 1 = f(1)$ and $f(1) \oplus 1 = f(0)$, therefore

$$|+\rangle|-\rangle = \frac{1}{2}(|0\rangle|0\rangle - |0\rangle|1\rangle + |1\rangle|0\rangle - |1\rangle|1\rangle) \to$$

$$\frac{1}{2}[|0\rangle|f(0)\rangle - |0\rangle|1 \oplus f(0)\rangle + |1\rangle|f(1)\rangle - |1\rangle|1 \oplus f(1)\rangle] =$$

$$= \frac{1}{2}[|0\rangle|f(0)\rangle - |0\rangle|f(1)\rangle + |1\rangle|f(1)\rangle - |1\rangle|f(0)\rangle] =$$

$$= \frac{1}{2}[|0\rangle - |1\rangle][|f(0)\rangle - |f(1)\rangle] = |-\rangle \left[\frac{|f(0)\rangle - |f(1)\rangle}{\sqrt{2}}\right]$$

After running $|+\rangle|-\rangle$ through the oracle, we may apply a Hadamard gate to the leftmost qubit and then measure it. This way we obtain $H|+\rangle = |0\rangle$ if f is constant and $H|-\rangle = |1\rangle$ if f is balanced.

A better way to understand the solution to Deutsch's problem is through *phase kickback*, the fact that controlled operations apply phase shifts to their control qubits. By considering it, we can simultaneously solve both cases in the problem statement:

$$U_f|+\rangle|-\rangle = \frac{1}{2}\Big(|0\rangle(|f(0)\oplus 0\rangle - |f(0)\oplus 1\rangle) + |1\rangle(|f(1)\oplus 0\rangle - |f(1)\oplus 1\rangle)\Big) =$$

$$= \frac{1}{2}\Big((-1)^{f(0)}|0\rangle(|0\rangle - |1\rangle) + (-1)^{f(1)}|1\rangle(|0\rangle - |1\rangle)\Big) =$$

$$= \frac{1}{\sqrt{2}}\Big((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\Big)|-\rangle =$$

$$= \frac{(-1)^{f(0)}}{\sqrt{2}}\Big(|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle\Big)|-\rangle$$

The modest quantum advantage from the Deutsch algortihm can be improved by keeping phase kickback in mind. In the Deutsch-Jozsa algorithm [1], the oracle now implements $f : \{0,1\}^n \to \{0,1\}$, which we are promised is either constant or balanced, and we are asked to classify it. We start with state $|0\rangle^{\otimes n}|1\rangle$, to which we apply a Hadamard transform:

$$H^{\otimes(n+1)}(|0\rangle^{\otimes n}|1\rangle) = \frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}|x\rangle|-\rangle$$

If we now run this state through the oracle, as before, we get

$$\frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}(-1)^{f(x)}|x\rangle|-\rangle$$

Next, we take the n control qubits $\frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}(-1)^{f(x)}|x\rangle$ and apply a Hadamard transform $H^{\otimes n}$ to them:

$$\frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\left[\frac{1}{2^{n/2}}\sum_{y=0}^{2^n-1}(-1)^{\sum_{i=1}^{n}x_iy_i}|y\rangle\right] = \sum_{y=0}^{2^n-1}\left[\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}(-1)^{\sum_{i=1}^{n}x_iy_i}\right]|y\rangle$$

Finally, we conclude that the probability to measure $|z\rangle$ is $\left|\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}(-1)^{\sum_{i=1}^{n}x_iz_i}\right|^2$, so

$$\text{Prob}(|0\rangle^{\otimes n}) = \left|\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\right|^2 = \begin{cases}1 \text{ if } f(x) \text{ is constant}\\ 0 \text{ if } f(x) \text{ is balanced}\end{cases}$$

thus solving the problem in a single evaluation of f. For a classical solution, we would have to check just over half of all possible bit strings ($2^{n-1}+1$), meaning the Deutsch-Jozsa algorithm has an exponential advantage over classical algorithms.

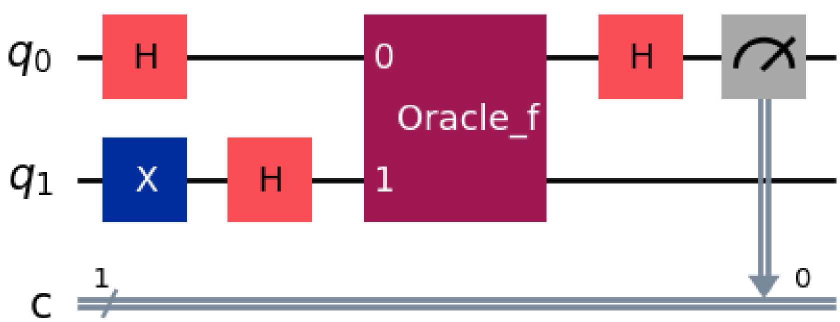
Question 6: Define a quantum circuit to implement Deutsch's algorithm with the function f and another with the function g.

Answer:

For the function f:

[10]:
```
deutsch_f = QuantumCircuit(2,1)
deutsch_f.x(1)
deutsch_f.h([0, 1])
deutsch_f.unitary(Oracle_f, [0, 1], label='Oracle_f')
deutsch_f.h(0)
deutsch_f.measure(0,0)
deutsch_f.draw(output='mpl')
```

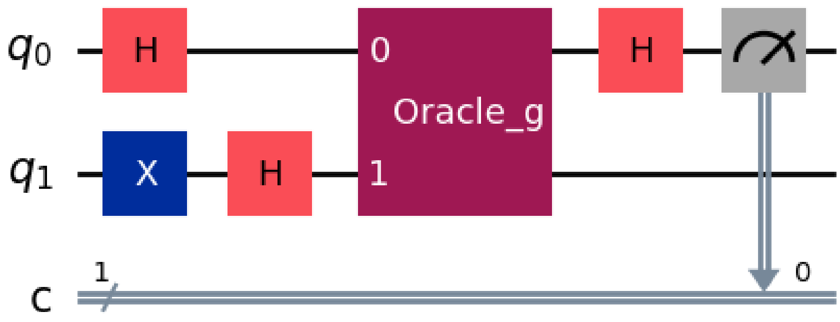
[10]:



And for the function g we just change the oracle:

[11]:
```
deutsch_g = QuantumCircuit(2,1)
deutsch_g.x(1)
deutsch_g.h([0, 1])
deutsch_g.unitary(Oracle_g, [0, 1], label='Oracle_g')
deutsch_g.h(0)
deutsch_g.measure(0,0)
deutsch_g.draw(output='mpl')
```

[11]:



[26]:
```
# Execute the circuit
job_Deutsch_S = sampler_S.run([deutsch_f, deutsch_g], shots=1)
result_Deutsch_S = job_Deutsch_S.result()

# Print the measurement results
if int(*result_Deutsch_S[0].data.c.get_counts()) == 0:
    print('f is constant')
else:
    print('f is balanced')
if int(*result_Deutsch_S[1].data.c.get_counts()) == 0:
    print('g is constant')
else:
    print('g is balanced')
```

```
f is balanced
g is constant
```

Question 7: Does the result of your quantum computation using Deutsch's algorithm agree with the previous classical result?

Answer: The classical result is that f is balanced and g is constant. After applying Deutsch's algorithm to both functions, we obtained the same result — that is, that f is balanced and g is constant. Our run in the lab with UCM tokens yielded the same result.

[13]:
```
# Transpile circuits to backend architecture
deutsch_f_t = transpile(deutsch_f, backend=backend_Q)
deutsch_g_t = transpile(deutsch_g, backend=backend_Q)

# Execute the circuit
job_Deutsch_Q = sampler_Q.run([deutsch_f_t, deutsch_g_t], shots=3)

# Save job identification code for later
df.deutsch_id = job_Deutsch_Q.job_id()
```

```python
df.to_csv('session.csv', index=False)
```

```
c:\Users\zapat\Escritorio\CODE\PIE_Compu_Cuantica\venv\Lib\site-
packages\qiskit_ibm_runtime\qiskit_runtime_service.py:879: UserWarning: Your
current pending jobs are estimated to consume 649.5556236231932 quantum seconds,
but you only have 588 quantum seconds left in your monthly quota; therefore, it
is likely this job will be canceled
  warnings.warn(warning_message)
```

[5]:
```python
# The following cell retrieves information about the job
job_Deutsch_Q = service.job(df.deutsch_id[0])
job_Deutsch_Q.status()
```

[5]: <JobStatus.DONE: 'job has successfully run'>

[6]:
```python
# The next statement can put your session on hold until the job runs and returns
result_Deutsch_Q = job_Deutsch_Q.result()
ans_f = result_Deutsch_Q[0].data.c.get_counts()
ans_g = result_Deutsch_Q[1].data.c.get_counts()

# Print the measurement results
if int(max(ans_f, key=ans_f.get)) == 0:
    print('f is constant')
else:
    print('f is balanced')
if int(max(ans_g, key=ans_g.get)) == 0:
    print('g is constant')
else:
    print('g is balanced')
```

```
f is balanced
g is constant
```

Experimental measurement of a Bell inequality (CHSH combination of correlators)

If $\{A_1, A_2\}$ and $\{B_1, B_2\}$ are two pairs of observables (with dichotomic/binary outcome) of two spatially separated systems, the expected values of their products $\langle A_i B_j \rangle$ according to any local hidden variable model (that is, an attempt at trying to explain away quantum features with additional classical mechanics variables) satisfy the classical CHSH inequality

$$|\langle A_1 B_1 \rangle + \langle A_1 B_2 \rangle + \langle A_2 B_1 \rangle - \langle A_2 B_2 \rangle| \leq 2.$$

Quantum theory, on the contrary, predicts that this inequality is violated for a suitable choice of observables, obtaining the maximum violation, in the case of two qubits, when

$$A_1 = X, \quad A_2 = Y, \quad B_1 = \frac{-(X+Y)}{\sqrt{2}}, \quad B_2 = \frac{-(X-Y)}{\sqrt{2}}$$

or rotationally equivalent configurations. Here X and Y denote the σ_x and σ_y Pauli matrices.

Bell inequality proof [2]:

Let's consider a system of two spin-$\frac{1}{2}$ particles which move in opposite directions towards observers Alice and Bob. The observers perform measurements A and B of the spin of their respective particles ($A, B = \pm 1$). The system is then described by the direction of A and B's Stern-Gerlach devices ($\mathbf{a}$, $\mathbf{b}$), as well as some local hidden variables λ (with probability density $\rho(\lambda)$) which seek to explain quantum correlations at long distances. Since A and B are sufficiently separated $A \neq A(\mathbf{b})$, $B \neq B(\mathbf{a})$:

$$A = A(\mathbf{a}, \lambda) \quad B = B(\mathbf{b}, \lambda)$$

We can be more general and average over hidden variables in the measurement instruments, obtaining

$$|\bar{A}| \leq 1 \quad |\bar{B}| \leq 1$$

Therefore the expected value of $\bar{A}\bar{B}$ (measurement correlation) in a particular set of directions is

$$E(\mathbf{a}, \mathbf{b}) = \int \mathrm{d}\lambda \rho(\lambda) \bar{A}(\mathbf{a}, \lambda) \bar{B}(\mathbf{b}, \lambda)$$

If we consider a different set of directions $\mathbf{a}'$, $\mathbf{b}'$

$$E(\mathbf{a}, \mathbf{b}) - E(\mathbf{a}, \mathbf{b}') =$$

$$= \int \mathrm{d}\lambda \rho(\lambda) \left[\bar{A}(\mathbf{a}, \lambda) \bar{B}(\mathbf{b}, \lambda) - \bar{A}(\mathbf{a}, \lambda) \bar{B}(\mathbf{b}', \lambda) \right] =$$

$$= \int \mathrm{d}\lambda \rho(\lambda) \left[\bar{A}(\mathbf{a}, \lambda) \bar{B}(\mathbf{b}, \lambda)(1 \pm \bar{A}(\mathbf{a}', \lambda) \bar{B}(\mathbf{b}', \lambda)) \right]$$

$$- \int \mathrm{d}\lambda \rho(\lambda) \left[\bar{A}(\mathbf{a}, \lambda) \bar{B}(\mathbf{b}', \lambda)(1 \pm \bar{A}(\mathbf{a}', \lambda) \bar{B}(\mathbf{b}, \lambda)) \right]$$

Applying the triangle inequality and considering $|\bar{A}| \leq 1$, $|\bar{B}| \leq 1$:

$$|E(\mathbf{a}, \mathbf{b}) - E(\mathbf{a}, \mathbf{b}')| \leq$$

$$\leq \left| \int \mathrm{d}\lambda \rho(\lambda) \left[\bar{A}(\mathbf{a}, \lambda) \bar{B}(\mathbf{b}, \lambda)(1 \pm \bar{A}(\mathbf{a}', \lambda) \bar{B}(\mathbf{b}', \lambda)) \right] \right|$$

$$+\left|\int \mathrm{d}\lambda\rho(\lambda)\left[\bar{A}(\mathbf{a},\lambda)\bar{B}(\mathbf{b}',\lambda)(1\pm\bar{A}(\mathbf{a}',\lambda)\bar{B}(\mathbf{b},\lambda))\right]\right| \leq$$

$$\leq \int \mathrm{d}\lambda\rho(\lambda)(1\pm\bar{A}(\mathbf{a}',\lambda)\bar{B}(\mathbf{b}',\lambda))$$

$$+\int \mathrm{d}\lambda\rho(\lambda)(1\pm\bar{A}(\mathbf{a}',\lambda)\bar{B}(\mathbf{b},\lambda)) =$$

$$= 2 \pm [E(\mathbf{a}',\mathbf{b}') + E(\mathbf{a}',\mathbf{b})]$$

So we obtain the following:

$$|E(\mathbf{a},\mathbf{b}) - E(\mathbf{a},\mathbf{b}')| \leq 2 \pm [E(\mathbf{a}',\mathbf{b}') + E(\mathbf{a}',\mathbf{b})] \leq 2 - |E(\mathbf{a}',\mathbf{b}') + E(\mathbf{a}',\mathbf{b})|$$

And finally applying the triangle inequality again

$$2 \geq |E(\mathbf{a},\mathbf{b}) - E(\mathbf{a},\mathbf{b}')| + |E(\mathbf{a}',\mathbf{b}') + E(\mathbf{a}',\mathbf{b})| \geq |E(\mathbf{a},\mathbf{b}) - E(\mathbf{a},\mathbf{b}') + E(\mathbf{a}',\mathbf{b}') + E(\mathbf{a}',\mathbf{b})|$$

If we label $A(\mathbf{a}) = A_2,\ A(\mathbf{a}') = A_1,\ B(\mathbf{b}) = B_1,\ B(\mathbf{b}') = B_2$ we recover the inequality stated above.

Maximisation:

We want to maximise $S \equiv E(\mathbf{a},\mathbf{b}) - E(\mathbf{a},\mathbf{b}') + E(\mathbf{a}',\mathbf{b}') + E(\mathbf{a}',\mathbf{b})$ in a state $|\psi\rangle$.

If we notice that

$$E(\mathbf{a},\mathbf{b}) = \langle\psi|(a^i\sigma_i)\otimes(b^j\sigma_j)|\psi\rangle = a^i\langle\psi|\sigma_i\otimes\sigma_j|\psi\rangle b^j$$

and call the correlation matrix $\langle\psi|\sigma_i\otimes\sigma_j|\psi\rangle = C_{ij}$, we can rewrite the CHSH correlator as

$$S = \mathbf{a}^{\mathrm{T}}C(\mathbf{b}-\mathbf{b}') + \mathbf{a}'^{\mathrm{T}}C(\mathbf{b}+\mathbf{b}')$$

A general optimisation is complicated, but we need only consider the singlet state $|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$, which we'll later use. In this state $C_{ij} = -\delta_{ij}$, therefore $E(\mathbf{a},\mathbf{b}) = -\mathbf{a}\cdot\mathbf{b} = -\cos\theta_{\mathbf{ab}}$:

$$\langle\psi|(a^i\sigma_i)\otimes(b^j\sigma_j)|\psi\rangle =$$

$$= \frac{1}{\sqrt{2}}\langle\Psi^-|\Big[(a_x|1\rangle + ia_y|1\rangle + a_z|0\rangle)(b_x|0\rangle - ib_y|0\rangle - b_z|1\rangle)$$

$$-(a_x|0\rangle - ia_y|0\rangle - a_z|1\rangle)(b_x|1\rangle + ib_y|1\rangle + b_z|0\rangle)\Big] =$$

$$= \frac{1}{\sqrt{2}}\langle\Psi^-|\Big[-(\mathbf{a}\cdot\mathbf{b})|01\rangle + (\mathbf{a}\cdot\mathbf{b})|10\rangle + \cdot\ |00\rangle + \cdot\ |11\rangle\Big] =$$

$$= -\mathbf{a}\cdot\mathbf{b}$$

Clearly S is maximised for

$$\begin{cases} (\mathbf{b}-\mathbf{b}') \ || -\mathbf{a} \\ (\mathbf{b}+\mathbf{b}') \ || -\mathbf{a}' \end{cases} \Rightarrow \begin{cases} \mathbf{b} = \frac{-(\mathbf{a}+\mathbf{a}')}{\|\mathbf{a}+\mathbf{a}'\|} \\ \mathbf{b}' = \frac{\mathbf{a}-\mathbf{a}'}{\|\mathbf{a}+\mathbf{a}'\|} \end{cases}$$

And we want $\|\mathbf{a}+\mathbf{a}'\| = \sqrt{2+2\cos\theta_{\mathbf{aa}'}}$ to be minimum, which is obtained with $\mathbf{a}\perp\mathbf{a}'$.

Finally, $S_{\max} = 2\sqrt{2} \approx 2.828 > 2$.

Question 2: Using this command, define the four product observables that appear in the CHSH inequality: A_1B_1, A_2B_1, A_1B_2, and A_2B_2:

Answer:

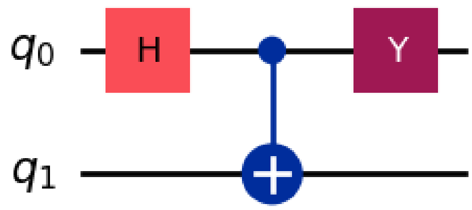
[4]:
```
# Define pairs of observables for maximum violation of the CHSH innequality
coef = 1 / np.sqrt(2)
A1B1aux = SparsePauliOp.from_list([('XX', - coef), ('XY', - coef)])
A1B2aux = SparsePauliOp.from_list([('XX', - coef), ('XY', + coef)])
A2B1aux = SparsePauliOp.from_list([('YX', - coef), ('YY', - coef)])
A2B2aux = SparsePauliOp.from_list([('YX', - coef), ('YY', + coef)])
Obsaux = [A1B1aux, A1B2aux, A2B1aux, A2B2aux]

# We need to fill the rest of the qubits with the identity (do nothing)
# this is thought for the current 127-qubit machines
n_remaining_qubits = 125
I125 = SparsePauliOp('I' * n_remaining_qubits)
Obs = [I125.tensor(op) for op in Obsaux]
```

Question 1: Define a circuit that prepares the "singlet state" of two qubits $|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$.

Answer: The following circuit specifically prepares the state $\frac{i}{\sqrt{2}}(|01\rangle - |10\rangle)$, which is physically equivalent to the requested state.

[5]:
```
# Prepare the input circuit:
chsh_circuit = QuantumCircuit(2)
chsh_circuit.h(0) # Hadamard on the first qubit
chsh_circuit.cx(0, 1) # NOT controlled to the first qubit
chsh_circuit.y([0]) # Y-Pauli gate to the first qubit
chsh_circuit.draw(output='mpl')
```

[5]:



Question 3: Calculate the simulated value obtained for the CHSH inequality jointly with its error.

Answer: The code we ran in the lab with UCM tokens yielded:

```
The simulated result is 2.81824 + 0.04:
```

```
does it exceed 2 with sufficient statistical certainty?
```

We can perform the calculations again ourselves.

[8]:
```
# Execute the circuit
job_CHSH_S = estimator_S.run([(chsh_circuit, Obsaux)])
result_S = job_CHSH_S.result()

# Statistical results
values_S = result_S[0].data.evs
Standard_errors_S = result_S[0].data.stds

# Print simulated expectation values
print('Simulated expectation values for the four correlators:')
print([f'{v:.3f} ± {s:.3f}' for v, s in zip(values_S, Standard_errors_S)])
```

```
Simulated expectation values for the four correlators:
['0.711 ± 0.011', '0.703 ± 0.011', '0.702 ± 0.011', '-0.713 ± 0.011']
```

[9]:
```
# CHSH value
CHSH_mean_S = abs(values_S[0] + values_S[1] + values_S[2] - values_S[3])
CHSH_uncertainty_S = sum(Standard_errors_S)
print(f'''
The simulated result is {CHSH_mean_S:.3f} ± {CHSH_uncertainty_S:.3f}:
does it exceed 2 with sufficient statistical certainty?
''')
```

```
The simulated result is 2.828 ± 0.044:
does it exceed 2 with sufficient statistical certainty?
```

Question 4: Compute, with a hand calculator or simple python commands, the experimental value obtained for the CHSH inequality jointly with its error from the quantum data obtained.

Answer: The code we ran in the lab with UCM tokens yielded:

`The simulated result is 2.60215 + 0.41467:`

`does it exceed 2 with sufficient statistical certainty?`

Let's perform the calculations again ourselves.

[65]:
```
# Transpile circuit to backend architecture
chsh_circuit_t = transpile(chsh_circuit, backend=backend_Q)

# Execute the circuit
job_CHSH_Q = estimator_Q.run([(chsh_circuit_t, Obs)])

# Save job identification code for later
df.chsh_id = job_CHSH_Q.job_id()
df.to_csv('session.csv', index=False)
```

```
c:\Users\zapat\Escritorio\CODE\PIE_Compu_Cuantica\venv\Lib\site-
packages\qiskit_ibm_runtime\qiskit_runtime_service.py:879: UserWarning: Your
current pending jobs are estimated to consume 623.1034379365619 quantum seconds,
but you only have 586 quantum seconds left in your monthly quota; therefore, it
is likely this job will be canceled
  warnings.warn(warning_message)
```

[7]:
```
# The following cell retrieves information about the job
job_CHSH_Q = service.job(df.chsh_id[0])
job_CHSH_Q.status()
```

[7]:
```
<JobStatus.DONE: 'job has successfully run'>
```

[8]:
```
# The next statement can put your session on hold until the job runs and returns
result_Q = job_CHSH_Q.result()

# Statistical results
values_Q = result_Q[0].data.evs
Standard_errors_Q = result_Q[0].data.stds

# Print simulated expectation values
print('Simulated expectation values for the four correlators:')
print([f'{v:.3f} ± {s:.3f}' for v, s in zip(values_Q, Standard_errors_Q)])
```

```
Simulated expectation values for the four correlators:
['0.689 ± 0.036', '0.801 ± 0.036', '0.744 ± 0.034', '-0.665 ± 0.034']
```

[9]:
```
# CHSH value
values_Q = result_Q[0].data.evs
Standard_errors_Q = result_Q[0].data.stds
```

```python
CHSH_mean_Q = abs(values_Q[0] + values_Q[1] + values_Q[2] - values_Q[3])
CHSH_uncertainty_Q = sum(Standard_errors_Q)
print(f'''
The simulated result is {CHSH_mean_Q:.3f} ± {CHSH_uncertainty_Q:.3f}:
does it exceed 2 with sufficient statistical certainty?
''')
```

```
The simulated result is 2.899 ± 0.139:
does it exceed 2 with sufficient statistical certainty?
```

The qubits we used for our computation exhibit quantum statistics, and thus are properly entangled. A computation such as this serves just as much as a proof of CHSH violation as it does as a calibration of a quantum computer.

References

[1] Cleve, R., Ekert, A., Macchiavello, C., & Mosca, M. (1998). Quantum algorithms revisited. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, 454(1969), 339–354*. Retrieved from https://arxiv.org/abs/quant-ph/9708016.

[2] Bell, J. S. (1971). *Introduction to the hidden-variable question* (No. CERN-TH-1220). CM-P00058691. Retireved from https://cds.cern.ch/record/400330/files/CM-P00058691.pdf.