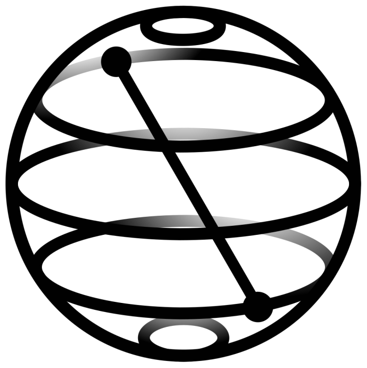
Introduction to Quantum Programming

Ugo Nzongani

Contents



1 Introduction

1.1 Libraries

Quantum programming can be done using several programming libraries, we are going to use Qiskit which is a Python library for quantum computing [1]. With Qiskit we can run our code on IBM real quantum devices. It is strongly advised to install a specific virtual environment. Here are the steps to do it:

```
python3 -m venv my_env
source my_env/bin/activate
```

Then run the following commands:

```
pip install jupyter
pip install wheel
pip install ipykernel
pip install pylatexenc
```

All the necessary libraries are in `requirements.txt`, to install them run `pip install -r requirements.txt`. The imports and functions are in `utils.py`, they can be imported in your file with the line `from utils import *`. One can leave the virtual environment with the command `deactivate`.

1.2 Introduction to Qiskit

1.2.1 Quantum circuit

A circuit in Qiskit in composed of quantum and classical registers. The quantum registers are used to perform the quantum computations, and the classical ones to store the classical outcome of the measurements. In Qiskit a register is always initialize in state $|0\rangle$.

```
n = 3
m = n
# Quantum register of n qubits
q = QuantumRegister(n, name='q')
# Classical register of m bits
c = ClassicalRegister(m, name='c')
# Quantum circuit
qc = QuantumCircuit(q,c)
# Apply X on qubit 0
qc.x(q[0])
# Apply H on qubit 0
qc.h(q[0])
# Apply a CNOT on qubit 0 controlled by qubit 2
qc.cx(q[2],q[0])
# Put a barrier between the operations
qc.barrier()
# Measure of the register q, the results are stored in c
qc.measure(q,c)
# Drawing the circuit
qc.draw('mpl')
# Note that the display style of the circuit can be modified with:
# qc.draw('mpl',style'bw'), qc.draw('mpl',style'iqx') or qc.draw()
```

Once a quantum circuit is created, one can run it to get some outcomes.

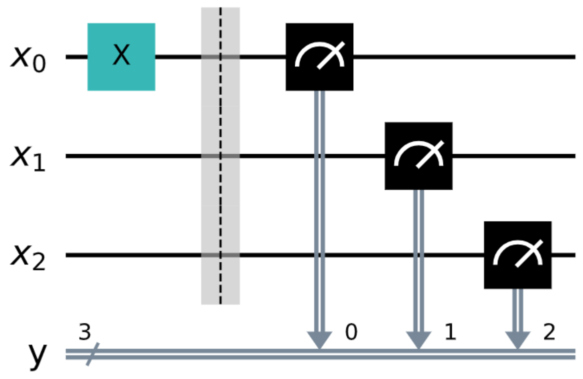
```
simulator = AerSimulator()
result = simulator.run(transpile(qc, simulator),shots=1024).result()
counts = result.get_counts()

# Plot the results of the measurements
plot_histogram(counts)
```

The variable `counts` is a dictionary that contains the results of the measurements. The keys are the bit string measured and the values corresponds to the number of times this string has been measured. Hence, the dictionary `{'00':134,'01':237,'10':511,'11':118}` indicates that we measured the state $|00\rangle$ 134 times, $|01\rangle$ 237 times, $|10\rangle$ 511 times and $|11\rangle$ 118 times.

It is also possible to only measure some qubits of the system:

```
# Measure of the qubit 0 of the register q
qc.measure(q[0],c[0])
```

The top wire corresponds to the qubit encoding the least significant bit. Let $|x_2x_1x_0\rangle = |000\rangle$ be a quantum state, we want to apply a NOT gate on $|x_0\rangle$, its Qiskit circuit is:



The outcome of the measurement is 001 with probability 1.

1.2.2 Sub-circuit

It is possible to transform a quantum circuit into a unitary gate, one can see it as a subroutine. Let's create a simple circuit:

```
q = QuantumRegister(2,name='q')
qc = QuantumCircuit(q)
qc.cx(q[0],q[1])
qc.h(q)
```

We transform it into a unitary gate:

```
gate = qc.to_gate(label='gate')
```

We can now add it as a subroutine in other quantum circuits:

```
q = QuantumRegister(4,name='q')
qc = QuantumCircuit(q)
qc.x(q)
# We apply the gate on the qubits 1 and 2 of the new circuit
qc.append(gate,[q[1],q[2]])
# We apply the same gate on the qubits 1 and 2 of the new circuit but controlled by qubit 0
qc.append(gate.control(),[q[0],q[1],q[2]])
qc.draw('mpl')
```

Then, we can decompose the boxes we just created:

```
qc.decompose().draw('mpl')
```

To increase the level of the decomposition, the parameter `reps` can be used:

```
qc.decompose(reps=1).draw('mpl')
```

We can compute the inverse, power or control gate easily:

```
q = QuantumRegister(5,name='q')
qc = QuantumCircuit(q)
# We apply the gate^2 on the qubits 1 and 2
qc.append(gate.power(2),[q[1],q[2]])
# We apply the inverse of the gate on the qubits 1 and 2
qc.append(gate.inverse(),[q[1],q[2]])
# We apply the gate on the qubits 2 and 3 controlled by qubits 0 and 1
qc.append(gate.control().control(),[q[0],q[1],q[2],q[3]])
qc.draw('mpl')
```

1.2.3 From unitary to gate

We can convert a unitary matrix into a quantum gate, let's create the gate $U = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/7} \end{pmatrix}$:

```
q = QuantumRegister(2,name='q')
qc = QuantumCircuit(q)
U = np.array([[1,0],[0,np.exp(1j*np.pi/7)]])
op = Operator(U)
qc.unitary(op,[q[0]],label='U')
qc.draw('mpl')
```

1.2.4 Run on IBM quantum device

To run your code on IBM real quantum computer, you first have to create an account on IBM Quantum Experience to get your API Key. Once it is done, you can run the following code:

```
# Bell quantum circuit
q = QuantumRegister(2, name='q')
c = ClassicalRegister(2, name='c')
qc = QuantumCircuit(q, c, name='Bell')
qc.h(q[0])
qc.cx(q[0], q[1])
qc.measure(q, c)

# Running it on IBM real device #

token = 'your_token'

# Show the available IBM devices
devices(token)

# Run the circuit on the specified device, if no device is given the circuit is run on the least busy one
counts = run_ibm(qc,token,device=None)
print(counts)
```

1.3 Usefull functions

One is encouraged to use the following functions defined in `utils.py` to run its circuit on a simulator and to plot quantum states:

- You can run a quantum circuit on a simulator with `run(qc,n_shots=1024,plot=True)`, the parameters `n_shots` and `plot` are optional. The first indicates the number of execution of the circuit[1], if the second is set to `False` the function returns a dictionnary containing the results of the execution.

[1] As the measure operation is probabilistic one has to execute a circuit several times to obtain a reliable probability distribution.

- You can plot a quantum state with `draw_state(qc,latex=False)`, the parameter `latex` is optional and if set to `True` the function displays the mathematical expression of the quantum state, otherwise the outcome is the Bloch Sphere of the state.

2 First quantum circuits

1. Plot the Bloch Sphere representation of the states $|0\rangle$,$|1\rangle$, $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Here is the code for $|0\rangle$:

```
q = QuantumRegister(1)
qc = QuantumCircuit(q)
bloch(qc,latex=False)
```

   If `latex` is set to `False` the output is the Bloch Sphere representation of the qubit state, otherwise it is the mathematical expression.

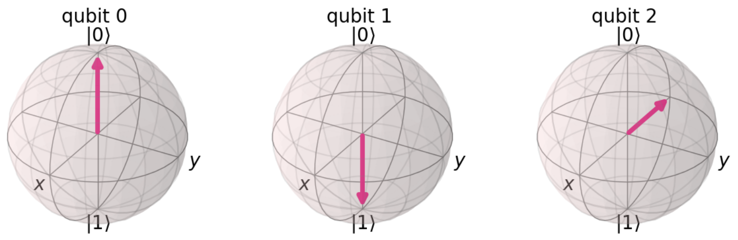
2. Do the same with the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)$.

3. Lastly, do it for the state $|\Psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. Can you explain what is happening ?

4. Design a quantum circuit that creates the state $|\psi\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$, run it and plot the measurement results, or display its statevector.

5. Design a quantum circuit that creates the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$, run it and plot the measurement results, or display its statevector.

6. Design a quantum circuit that creates the state $|\psi\rangle = \frac{i}{\sqrt{2}}(|010\rangle - |110\rangle)$[2] and display its statevector. Note that its Bloch Sphere representation is:



7. Using the function `random_unitary`, create a random state $|\psi\rangle$. Design a quantum circuit working on the state $|\psi\rangle \otimes |0\rangle$ that swaps these two qubits into:

$$|\psi\rangle \otimes |0\rangle \mapsto |0\rangle \otimes |\psi\rangle$$

   Plot its Bloch Sphere representation before and after to check the result of the computation.

8. One wishes to design a quantum circuit that performs the following transformation on n qubits:

$$\sum_{k=0}^{2^n-1} \alpha_k |k\rangle \mapsto \sum_{k=0}^{2^n-1} \alpha_k |2^n - 1 - k\rangle$$

   Note that it should work for any $(\alpha_0, \ldots, \alpha_{2^n-1}) \in \mathbb{C}^{2^n}$, not only for $\alpha_k = \frac{1}{\sqrt{2^n}}$. This transformation is just a permutation of the amplitudes α_k over the basis states, creating some kind of "mirror quantum state". Let's take an example with $n = 2$ qubits, in such a case this mapping acts as:

$$\begin{aligned} \alpha_0 |00\rangle &\mapsto \alpha_0 |11\rangle \\ \alpha_1 |01\rangle &\mapsto \alpha_1 |10\rangle \\ \alpha_2 |10\rangle &\mapsto \alpha_2 |01\rangle \\ \alpha_3 |11\rangle &\mapsto \alpha_3 |00\rangle \end{aligned}$$

   (a) Make a list `unitaries` of length n that contains random 2×2 unitaries.

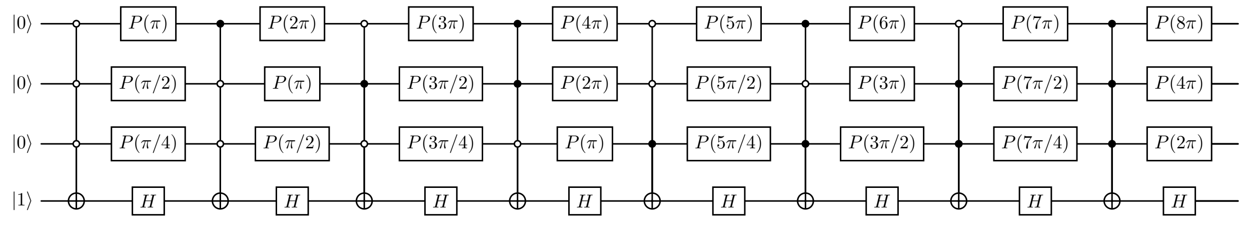
[2] *Hint: you only have to use X, Y and H gates.*

(b) Design a quantum circuit `qc1` that creates the state $\sum_{k=0}^{2^n-1} \alpha_k |k\rangle$, where the α_k are that of `unitaries`.

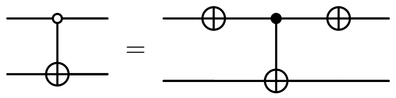
(c) Design a quantum circuit `qc2` that creates the state $\sum_{k=0}^{2^n-1} \alpha_k |2^n - 1 - k\rangle$, where the α_k are that of `unitaries`.

(d) Run the quantum circuits and plot their results, both plots should mirror one another.
It is possible to plot results from 2 quantum circuit at once with:
`plot_histogram([run(qc1, plot=False),run(qc2, plot=False)])`

9. Reproduce the following quantum circuit, try to do it using as little code as possible:



The phase gate $P(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$ can be used with `qc.p(theta,qubit)`, the multi-controlled NOT-gate can be used with `qc.mcx`, one is encouraged to check its documentation here. Lastly, one should remember the identity:



10. Can you generalize the previous circuit for $n > 1$ qubits knowing that it was $n = 3$?

3 Cloning qubits ?

In quantum computing it is not possible to copy an arbitrary qubit state $\alpha|0\rangle + \beta|1\rangle$ due to the No-Cloning theorem. Hence, such a transformation is impossible for arbitrary $|\psi\rangle$:

$$|\psi\rangle|0\rangle \mapsto |\psi\rangle|\psi\rangle$$

However, it does not mean that any kind of copy is impossible, let's try some experiments. Unknown quantum states can be prepared using the function `random_unitary`.

1. Given a quantum state $|a\rangle = |0\rangle$ or $|a\rangle = |1\rangle$, design a quantum circuit that performs the following computation:

$$|a\rangle \otimes |0\rangle \mapsto |a\rangle \otimes |a\rangle$$

2. Can you still find a circuit to do it when the second qubit is initialized to $|1\rangle$?

$$|a\rangle \otimes |1\rangle \mapsto |a\rangle \otimes |a\rangle$$

3. Given a quantum state $|a = a_n \dots a_1\rangle = |a_n\rangle \otimes \dots \otimes |a_1\rangle$ with $a_i \in \{0, 1\}$, design a quantum circuit that performs the following computation:

$$|a_n \dots a_1\rangle \otimes |0\rangle^{\otimes n} \mapsto |a_n \dots a_1\rangle \otimes |a_n \dots a_1\rangle$$

4. In questions (1) and (2), we made quantum circuits that copy the bit value of a qubit into an ancilla qubit. We now wish to do the same operation for n ancilla qubits, that is:

$$|a\rangle \otimes |0\rangle^{\otimes n} \mapsto |a\rangle \otimes |a\rangle^{\otimes n}$$

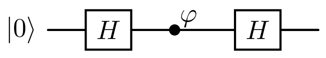
(a) Design a quantum circuit that performs this operation for $n \in \mathbb{N}^*$.

(b) The quantum circuit you just made probably has a time complexity of $O(n)$, i.e. a linear scaling with the number of ancilla qubits. Can you improve it to obtain a logarithmic scaling, i.e. $O(\log_2 n)$?

(c) Plot the quantum circuit execution time evolution as a function of the number of ancilla qubits, for the linear and logarithmic methods. The time execution can be obtained with `qc.depth()`.

4 Single qubit interference

Interferences play a major role in quantum computation, by going into the Fourier basis (applying a Hadamard on a single qubit) and applying a phase it is possible to create an interesting interference scheme as depending on the value of the phase, the outcome of the measurement will change.



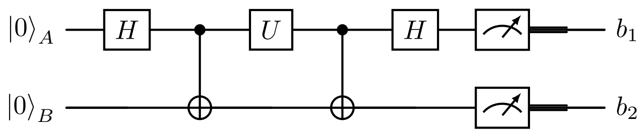
1. Implement the single qubit interference circuit with the phase gate $P(\varphi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix}$.

2. Plot the probabilities of measuring $|0\rangle$ and $|1\rangle$ as a function of φ, you can access the statevector `psi` of the circuit by adding the following lines at the end:

```
qc.save_statevector()
simulator = AerSimulator()
psi = simulator.run(qc).result().get_statevector(qc)
```

5 Superdense coding

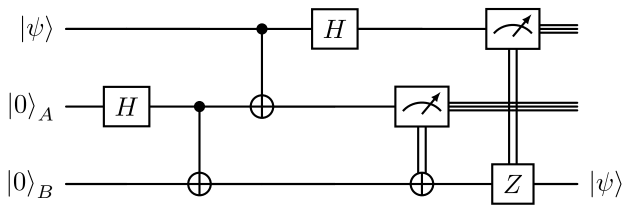
The Superdense coding protocol [2] corresponds to the following quantum circuit:



Note that the black box U corresponds to the operations performed by Alice depending on the informations she wants to send. Write a function `Superdense_Coding(b2,b1)` that takes the bits of information Alice wants to send to Bob as inputs, and implements the corresponding Superdense coding protocol.

6 Quantum Teleportation

The Quantum Teleportation protocol [3] is described with the quantum circuit:



1. Implement the Quantum Teleporation protocol.

2. Plot the Bloch Sphere of the 3 qubits system before and after teleportation. Do not forget to generate a random quantum state for the state $|\psi\rangle$ to teleport. Note that the state before teleportation is $|\psi\rangle \otimes \frac{1}{\sqrt{2}}(|0_A 0_B\rangle + |1_A 1_B\rangle)$.

7 Enumeration

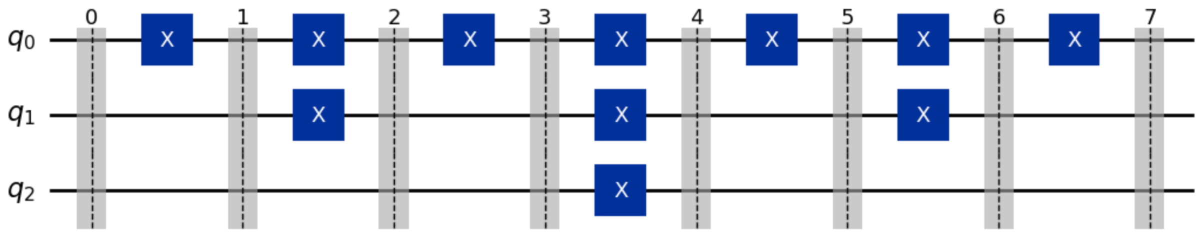
1. Write a function `counting(n)` that takes a number n of qubits as input and returns a quantum circuit that flip sequentially the qubits to generate the following integers:

$$|0\rangle \to |1\rangle \to |2\rangle \to \cdots \to |2^n - 1\rangle$$

Note that the notation $|2\rangle$ corresponds to the state $|10\rangle$ if working with 2 qubits, it would have been $|010\rangle$ for 3 qubits, etc... Use the line `qc.save_statevector(label=str(i))` to save the statevector of the qubits at iteration i. You can nicely visualize the different states of your circuit by running the code:

```
simulator = AerSimulator()
result = simulator.run(qc).result()
for i in range(2**n):
    psi = result.data(0)[str(i)]
    clear_output(wait=True)
    print('Bloch Sphere representation of '+str(i)+' in the computational basis:')
    display(psi.draw('latex'))
    display(plot_bloch_multivector(psi))
    time.sleep(3)
clear_output(wait=True)
```

You can uncomment the line `display(psi.draw('latex'))` to display the mathematical expression instead of the Bloch Sphere of the state. Here is the circuit you should get for $n = 3$:



2. The previous circuit is not using the optimal number of X gates because we generated the integers in increasing order. The optimal way of counting is to use Gray code. It is a binary representation of integers where two successive values differ in only one bit. Using this property, one is able to minimize the number of X gates. We show on Fig. 1 the 3-bits Gray code.

Decimal with Gray code ordering	*Binary representation*
0	000
1	001
3	011
2	010
6	110
7	111
5	101
4	100

Figure 1: 3-bits Gray code.

Write a function `gray_counting(n,gray_list)` that takes a number n of qubits, a list of Gray code bitstring of size 2^n, and returns a quantum circuit that generate the integers from 0 to $2^n - 1$ in Gray code order. Note that for $n = 3$ qubits, `gray_list` is: `['000', '001', '011', '010', '110', '111', '101', '100']`.

3. Plot the number of X gates as a function of the number of qubit n for `counting` and `gray_counting`.

8 Quantum Fourier Transform

The Quantum Fourier Transform (QFT) is the quantum analog of the Fourier Transform, it is used as a subroutine in many quantum algorithms. It acts on a quantum state of n qubits, with $N = 2^n$, as:

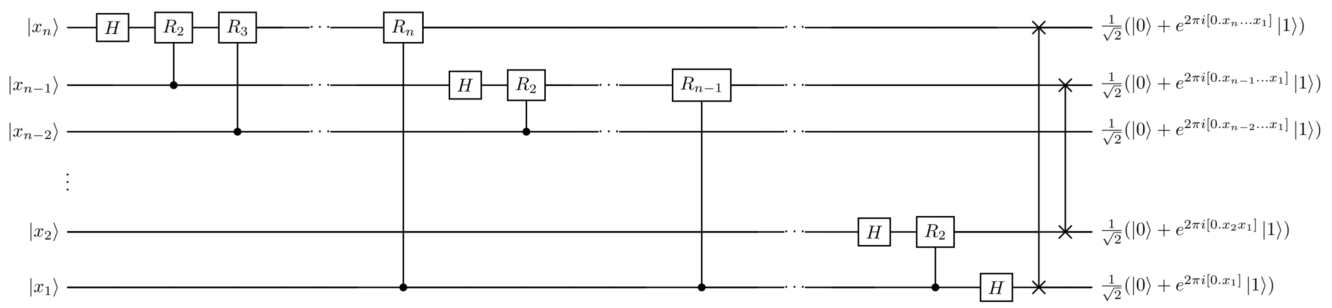
$$|x\rangle = \sum_{k=0}^{N-1} x_k |k\rangle$$

and maps it to

$$QFT\left|x\right\rangle = \sum_{k=0}^{N-1} y_k \left|k\right\rangle \text{ with } y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i \frac{jk}{N}}$$

The QFT is a transform between the computational basis and the Fourier basis. The computational basis encodes information in the $\left|k\right\rangle$ states unlike the Fourier basis that does it in the phases. For more informations, one is encouraged to click here.

The quantum circuit implementing the QFT is shown bellow with $R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}$:



$|x_n\rangle$
$|x_{n-1}\rangle$
$|x_{n-2}\rangle$
$|x_2\rangle$
$|x_1\rangle$
H
R_2
R_3
R_n
R_{n-1}
$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i[0.x_n \ldots x_1]}|1\rangle)$
$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i[0.x_{n-1} \ldots x_1]}|1\rangle)$
$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i[0.x_{n-2} \ldots x_1]}|1\rangle)$
$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i[0.x_2 x_1]}|1\rangle)$
$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i[0.x_1]}|1\rangle)$

1. Write a function `QFT(n,swaps=True)` that takes the number of qubits n and a boolean `swaps` as inputs, which implements the QFT. The boolean `swaps` indicates if one should include the swap step at the end of the circuit. The controlled-rotations can be implemented with `qc.cp(angle,control_qubit,target_qubit)`.

2. Run your function to generate the QFT circuit on 1 qubit. Did you expect this result?

3. In the previous exercice we enumerated integers from 0 to $2^n - 1$ in the computational basis, i.e. by flipping qubits with X gates. It turns out that we can do the same operations in the Fourier space by doing rotations instead of bit flips. Knowing that H is the QFT on 1-qubit, one should remember the identity:

$$X = HZH$$

with

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi} \end{pmatrix} = P(\theta = \pi)$$

More generally, one can go from basis state $\left|k\right\rangle$ to $\left|k+1 \mod 2^n\right\rangle$ using the following protocol:

   (a) Apply a QFT to go in Fourier basis (with no swaps)
   (b) For each qubit $j \in [0, n-1]$ apply a phase gate $P(\theta)$ with $\theta = 2\pi/2^{j+1}$
   (c) Apply the inverse QFT to go back to the computational basis (with no swaps)

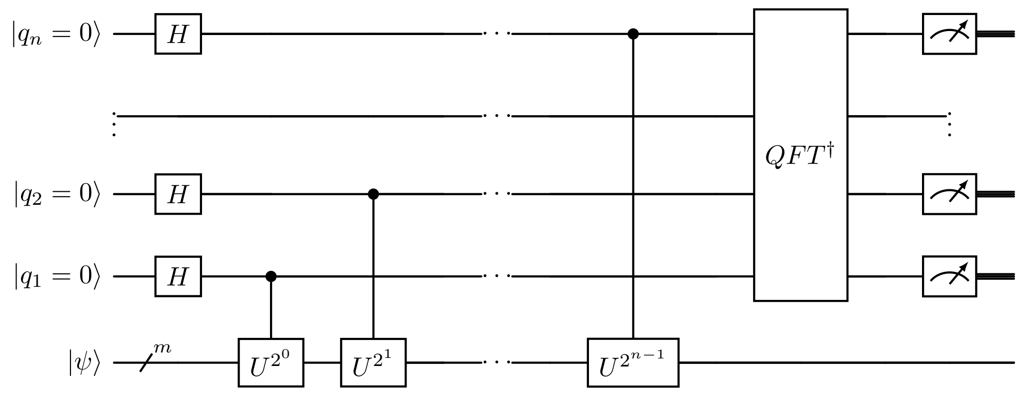
   Write a function `fourier_counting(n)` that takes a number n of qubits as input and returns a quantum circuit that sequentially counts integers $k \in [0, 2^n - 1]$ in Fourier space. Do not forget to retrieve the statevector of the state at each iteration k to visualize the modifications of the corresponding Bloch Spheres:

```
simulator = AerSimulator()
qc_transpile = transpile(qc, simulator)
result = simulator.run(qc_transpile).result()
for i in range(2**n):
    psi = result.data(0)[str(i)]
    clear_output(wait=True)
    print('Bloch Sphere representation of '+str(i)+' in the Fourier basis:')
    display(psi.draw('latex'))
    display(plot_bloch_multivector(psi))
    time.sleep(3)
clear_output(wait=True)
```

4. The complexity of a quantum circuit is usually defined by its number of 1 and 2 qubit gates. The complexity of the classical Fast Fourier Transform (FFT) is $O(N \log N)$ with $N = 2^n$. What is the asymptotic complexity of the Quantum Fourier Transform?

9 Quantum Phase Estimation

Quantum Phase Estimation (QPE) is a very usefull subroutine for finding the eigenvalues of a unitary U. As it is unitary, its eigenvalue are of the form $e^{2i\pi\varphi}$. Using QPE, one can find the phase φ as the outcome of the measurements is $\varphi 2^n$ with high probability. We recall that if $|\psi\rangle$ is an eigenvector of U, then $U|\psi\rangle = e^{2i\pi\varphi}|\psi\rangle$ for $\varphi \in \mathbb{R}$. It uses two quantum registers, the eigenvector $|\psi\rangle$ (acting on m qubits) and a working register of n qubits initialized to $|0\rangle$. Here is the quantum circuit implementation of QPE:



1. Implement the QPE circuit with $m = 1$ by writing a function `QPE(n,psi,U)` that takes the number of qubits n initialized to $|0\rangle$, `psi` a unitary to initialize the state of the bottom qubit, and a unitary U as inputs.

2. To check if your implementation is correct, we are going to estimate the value of π using QPE. Let $|\psi\rangle$ be an eigenvector of U, therefore $U|\psi\rangle = e^{2i\pi\varphi}|\psi\rangle$. Using QPE we can easily get φ since $\varphi = \frac{\delta}{2^n}$, where δ is the outcome of the measurement. Let's pick U such that $U = \begin{pmatrix} 1 & 0 \\ 0 & e^i \end{pmatrix}$. Hence, we have $U|1\rangle = e^i|1\rangle$, with $|1\rangle$ and e^i being eigenvector and eigenvalue of U. We recall that any eigenvalue of a unitary matrix can be written as $e^{2i\pi\varphi}$, thus $2\pi\varphi = 1$, i.e. $\pi = \frac{1}{2\varphi}$.
   The protocol is summarized bellow:

   - Compute the QPE circuit with $|\psi\rangle = |1\rangle$ and $U = \begin{pmatrix} 1 & 0 \\ 0 & e^i \end{pmatrix}$
   - Get the outcome δ by measuring the n qubit register
   - Compute $\pi = \frac{1}{2\frac{\delta}{2^n}}$

   (a) Write a function `estimate_pi(n)` that uses QPE to return an estimation of π.

   (b) Plot the quantum estimation of π as a function of the number of qubits $n \in [2, \ldots, 15]$.

10 Quantum Addition: Draper Adder

With a quantum computer it is possible to perform additions by using the QFT [4]. Given two quantum register of n qubits each, $|a\rangle$ and $|b\rangle$, we wish to compute the sum of them and to store the result in $|b\rangle$. Let's see how this algorithm works. Initially the state of the system is:

$$|\psi\rangle = |a\rangle \otimes |b\rangle$$

After applying the QFT on the second register it becomes:

$$\begin{aligned}|\psi\rangle &= |a\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.b_n \ldots b_1)}|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.b_{n-1} \ldots b_1)}|1\rangle) \otimes \cdots \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.b_1)}|1\rangle) \\ &= |a\rangle \otimes |\phi_n(b)\rangle \otimes |\phi_{n-1}(b)\rangle \otimes \cdots \otimes |\phi_1(b)\rangle\end{aligned}$$

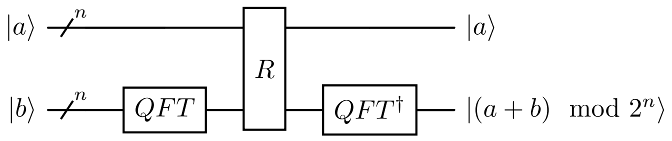
Then, we perform some rotations on $|b\rangle$ controlled by $|a\rangle$:

$$\begin{aligned}|\psi\rangle &= |a\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.a_n\ldots a_1+0.b_n\ldots b_1)}|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.a_{n-1}\ldots a_1+0.b_{n-1}\ldots b_1)}|1\rangle) \otimes \cdots \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.a_1+0.b_1)}|1\rangle)\\ &= |a\rangle \otimes |\phi_n(a+b)\rangle \otimes |\phi_{n-1}(a+b)\rangle \otimes \cdots \otimes |\phi_1(a+b)\rangle\end{aligned}$$

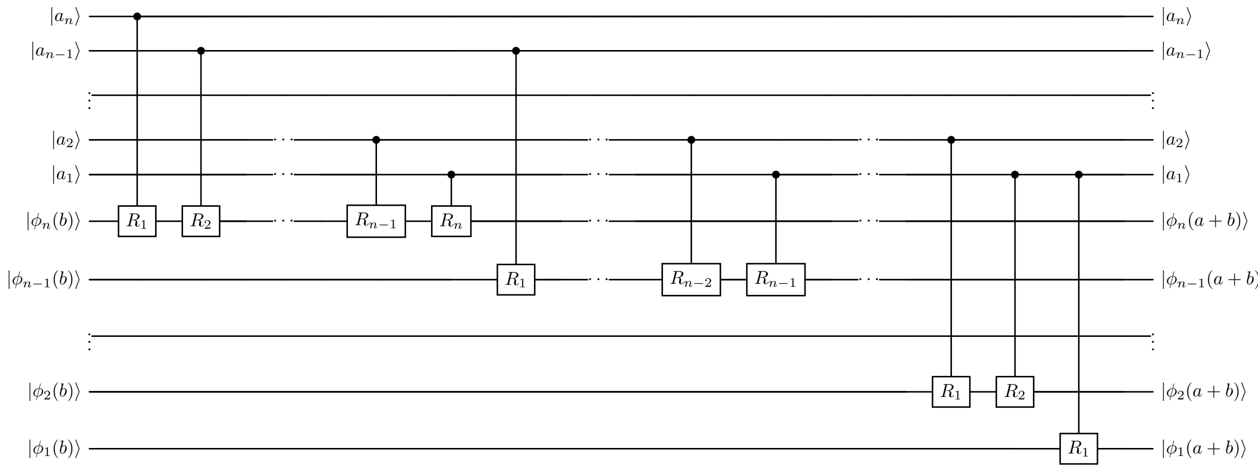
Finally, as the information is now encoded in the phases, we go back to the computational basis by applying the inverse QFT on $|b\rangle$:

$$\begin{aligned}|\psi\rangle &= |a\rangle \otimes |(a+b)_n\rangle \otimes |(a+b)_{n-1}\rangle \otimes \cdots \otimes |(a+b)_1\rangle\\ &= |a\rangle \otimes |(a+b) \mod 2^n\rangle\end{aligned}$$

The quantum circuit representation of the algorithm is shown bellow:



The controlled-rotations R step is shown bellow with $R_k = \begin{pmatrix}1 & 0\\ 0 & e^{2\pi i/2^k}\end{pmatrix}$:



Implement the Draper Adder. **To make it work you must not do the swaps in the QFT.** Therefore, you have to set `swaps=False` when calling the `QFT` function. We will do the implementation in two steps:

- The initialization of the registers $|a\rangle$ and $|b\rangle$.
- The implementation of the Draper Adder.

1. Write a function `Draper_Adder(x,y)` that takes two integers x and y as inputs and compute their sum using the Draper Adder. The first quantum register $|a\rangle$ is going to store the value of x and $|b\rangle$ the value of y. Start by initializing the two quantum registers correctly with the corresponding NOT gates. We recall that $|x = a_n \ldots a_1\rangle$ and $|y = b_n \ldots b_1\rangle$ where the index 1 corresponds to the least significant qubit. Then, implement the Draper Adder with no swaps for the QFT steps. Note that your function should work for any arbitrary non negative integers values of x and y.

   You can use the line `bin(i)[2:].zfill(n)` to get the binary string representation of integer i on n bits, setting n to 0 gives the binary representation of i with the minimum number of bits. Recall that in Python the first element $s[0]$ of a bit string s is the leftmost bit, you can reverse a string s with `reversed(s)` or `s = s[::-1]`.

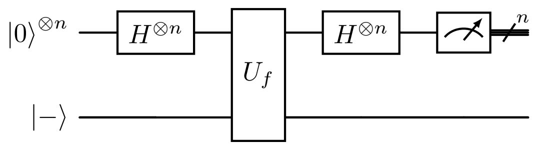
2. Can you modify your function so that it can compute $y - x$ instead of $x + y$?

You can verify the correctness of your implementation with the function `draper_verification(qc,x,y,sub)`, where `sub` is a boolean set to `False` if your circuit computes the subtraction $y - x$.

11 Bernstein-Vazirani Algorithm

In the Bernstein-Vazirani problem, one has to guess a secret number encoded on n bits. Classicaly the most efficient method would require n trials. Indeed, one would first do the logical operation AND between 1 and the first bit, if this bit is 1 the result would be 1, otherwise 0. Repeating this procedure for the n bits, one is able to guess the secret number with a time complexity $O(n)$. However, with a quantum computer one can solve this problem with only 1 trial using Bernstein-Vazirani Algorithm [5], hence with a constant complexity $O(1)$. Let's formalize the problem.

Given a function $f : \{0,1\}^n \to \{0,1\}$ acting as an oracle where $f(x)$ is a scalar product between x and the secret bit string $s \in \{0,1\}^n$ modulo 2, with $f(x) = x \cdot s = x_1 s_1 \oplus \cdots \oplus x_n s_n$. The problem is to find the value of s. The quantum circuit implementing Bernstein-Vazirani Algorithm is shown bellow, where $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$:



Let's compute the effect of the algorithm on an input state. We start with the initial state and we only focus on the n qubits of input:

$$|\psi\rangle = |0\rangle^{\otimes n}$$

After applying the Hadamard tower on the first n qubits, we obtain:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$

The oracle U_f acts on $|x\rangle$ as $U_f |x\rangle = (-1)^{f(x)} |x\rangle$. The superposition becomes:

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

Reapplying the Hadamard tower, we get:

$$\begin{aligned} |\psi\rangle &= \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)+x\cdot y} |y\rangle \\ &= |s\rangle \end{aligned}$$

The final state is $|s\rangle$ because for a given value of y we have:

$$\begin{aligned} |\psi\rangle &= \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{f(x)+x\cdot y} \\ &= \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot s + x\cdot y} \\ &= \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot(s\oplus y)} \\ &= 1 \text{ if } s\oplus y = 0, 0 \text{ otherwise} \end{aligned}$$

As $s \oplus y = 0$ only if $s = y$, the only non zero amplitude is that of $|s\rangle$.

1. Write a function that generates a random bit string of length n.
2. Let's implement the oracle U_f. To do so we are going to use sequence of CNOT gates. Given a secret bit string $s = s_n \dots s_1$, one aims to apply a CNOT gate on the last qubit $|-\rangle$ controlled by $|x_i\rangle$ if $s_i = 1$. Recall that in Python the first element $s[0]$ of a bit string s is the leftmost bit, you can reverse a string s with `reversed(s)` or `s = s[::-1]`. As we do not use this convention in our encoding, do not forget to reverse the bit string s before entering the loop.

3. Merging everything together, write a function `Bernstein_Vazirani(n)` that takes a number n as input and implements the Bernstein-Vazirani Algorithm.
4. Run your code on IBM real quantum device.

12 Grover's Algorithm

Grover's algorithm [6] is one of the most famous quantum algorithms and is used as a subroutine in a lot of other algorithms. Its purpose is to solve unstructured search problems, which consist in finding a marked element in an unstructured database. Classically, searching an element in an unstructured database of N elements would take $O(N)$ operations, as in the worst case scenario one has to examine all the elements before finding the wanted one. However, on a quantum computer, such a task is feasible with only $O(\sqrt{N})$ operations, providing a quadratic speedup over classical computers. The heart of Grover's algorithm is to mark the element we wish to find with an oracle, and to amplify the probability of measuring it.

Given a set $S = \{0, 1, \dots, N-1\}$, the search problem can be formally defined as:

$$f(x) = \begin{cases} 1 \text{ if } x \text{ is the marked element} \\ 0 \text{ otherwise} \end{cases}$$

with $x \in S$. In order to solve the problem, one has to find x such that $f(x) = 1$. The algorithm works for multiples marked elements, however, we will focus on the single marked case. As we are working with qubits, it is convenient to work with databases of size $N = 2^n$. Each element of the set S corresponds to a computational basis state $|x\rangle$. The initial state is the uniform superposition as it corresponds to a fair random guess in the database:

$$H^{\otimes n} |0^n\rangle = \frac{1}{\sqrt{N}} \sum_{x \in S} |x\rangle \tag{1}$$

The first step of Grover's algorithm consists in marking the element, to do so we just use an oracle O_f that puts a negative phase in front of its corresponding basis state:

$$O_f |x\rangle = (-1)^{f(x)} |x\rangle = \begin{cases} |x\rangle \text{ if } f(x) = 0 \\ -|x\rangle \text{ if } f(x) = 1 \end{cases}$$

To simplify the circuit implementation of this step, it is usual to add an ancilla qubit in the state $|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$. Thus, one needs $n+1$ qubits to implement this algorithm, note that we only use the ancilla for the marking step. A controlled-phase flip is applied an the ancilla, if $f(x) = 1$ then $|-\rangle$ becomes $-|-\rangle = \frac{-|0\rangle + |1\rangle}{\sqrt{2}}$. Therefore, after applying the oracle, the state becomes:

$$\begin{aligned} O_f \left(\frac{1}{\sqrt{N}} \sum_{x \in S} |x\rangle \otimes |-\rangle \right) &= O_f \left(\frac{1}{\sqrt{N}} \sum_{x \in S, f(x)=0} |x\rangle \otimes |-\rangle + \frac{1}{\sqrt{N}} \sum_{x \in S, f(x)=1} |x\rangle \otimes |-\rangle \right) \\ &= \frac{1}{\sqrt{N}} \sum_{x \in S, f(x)=0} |x\rangle \otimes (-1)^{f(x)} |-\rangle + \frac{1}{\sqrt{N}} \sum_{x \in S, f(x)=1} |x\rangle \otimes (-1)^{f(x)} |-\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{x \in S, f(x)=0} |x\rangle \otimes |-\rangle + \frac{1}{\sqrt{N}} \sum_{x \in S, f(x)=1} |x\rangle \otimes -|-\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{x \in S, f(x)=0} |x\rangle \otimes |-\rangle + \frac{1}{\sqrt{N}} \sum_{x \in S, f(x)=1} -|x\rangle \otimes |-\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{x \in S} (-1)^{f(x)} |x\rangle \otimes |-\rangle \end{aligned}$$

Notice that the ancilla remains in state $|-\rangle$ after applying the oracle.

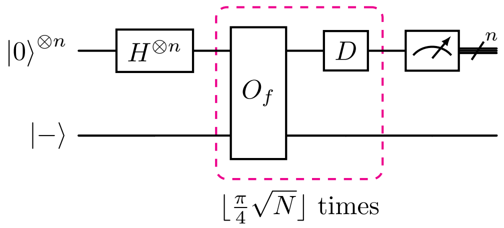
The second step is called the diffusion operator, it's a reflection over the average amplitude:

$$D = H^{\otimes n} \left(2 |0^n\rangle \langle 0^n| - I \right) H^{\otimes n}$$

where I is the identity and $2\left|0^n\right\rangle\left\langle 0^n\right| - I$ flips the phase of the $\left|O^n\right\rangle$ state. This operator is going to flip the states around the mean, as the sign of the marked element is different from the rest, its amplitude will grow and that of the unmarked states will decrease. Therefore, one has to repeat these operations k times to reach a probability of measuring the marked state close to 1. Grover's algorithm corresponds to applying the following operator:

$$G_f = ((D \otimes I) \cdot O_f)^k \left|+^n\right\rangle \otimes \left|-\right\rangle$$

The quantum circuit representation is:



Let us now derive the optimal number k of iteration. We first rewrite Eq. (1) as

$$|\psi\rangle = \frac{\sqrt{N-1}}{\sqrt{N}}\,|\alpha\rangle + \frac{1}{\sqrt{N}}\,|\beta\rangle = \cos\theta\,|\alpha\rangle + \sin\theta\,|\beta\rangle \tag{2}$$

with $|\alpha\rangle = \frac{1}{\sqrt{N-1}}\sum_{x\in S, f(x)=0}|x\rangle$ and $|\beta\rangle$ is the marked element. Notice from Eq. (2), that $\theta = \arcsin\frac{1}{\sqrt{N}}$. The state $|\alpha\rangle$ then corresponds to the unmarked states and $|\beta\rangle$ to the marked one. Each Grover iteration applies a rotation of angle 2θ in the subspace spanned by $|\alpha\rangle$ and $|\beta\rangle$. After k iterations we obtain the state $|\psi_k\rangle$, the angle between $|\psi_k\rangle$ and $|\alpha\rangle$ is $(2k+1)\theta$, thus:

$$|\psi_k\rangle = \cos((2k+1)\theta)\,|\alpha\rangle + \sin((2k+1)\theta)\,|\beta\rangle$$

We want to maximize the probability of success, therefore we want $(2k+1)\theta$ to be as close as possible to $\pi/2$:

$$\begin{aligned}(2k+1)\theta &= \frac{\pi}{2}\\ k &= \frac{\pi}{4\theta} - \frac{1}{2}\end{aligned}$$

Using the approximation $\arcsin\theta \approx \theta$:

$$k^* = \lfloor \frac{\pi}{4}\sqrt{N} - \frac{1}{2} \rfloor \approx \lfloor \frac{\pi}{4}\sqrt{N} \rfloor = O(\sqrt{N})$$

1. Write a function `grover(n,target_state)` that takes a number n of qubits and a bitstring representing the target state as inputs, and returns a quantum circuit implementing Grover's algorithm. You should do this implementation step by step:
   (a) Initialize the register of n qubits to the uniform super position and the ancilla to $|-\rangle$
   (b) Loop over the optimal number of iterations $k^* = \lfloor \frac{\pi}{4}\sqrt{N} \rfloor$:
      i. Apply the oracle O_f that is a multicontrolled-X gate whose control qubits are the first n and the target qubit is the ancilla. Note that the state of the control qubits must be that of the target state. As an example, if $n = 3$ and the target state is $|110\rangle$, the control qubits must be in state $|1\rangle, |1\rangle$ and $|0\rangle$. To implement the oracle you should use `qc.mcx`, feel free to check its documentation here.
      ii. Apply the diffusion operator which is composed of:
         A. A layer of Hadamard gates on the first n qubits.
         B. A multicontrolled-X gate whose control qubits are the first n and the target qubit is the ancilla. The state of the control qubits must be $|0^n\rangle$.
         C. Another layer of Hadamard gates on the first n qubits.

   Do not forget to add a classical register to measure the register of n qubits. Run your circuit for different values of n and several target states, you should obtain a success probability close to 1.

2. Write a function `grover_success(n,target_state,max_iter)` that takes a number n of qubits, a bitstring representing the target state and a maximum number of iterations as inputs. This function will return the same Grover circuit except that you will not put a classical register, not measure the qubits and set the number of iterations to `max_iter`. Retrieve the statevector at each iteration and plot the success probability as a function of the number of iterations. Note that as your circuit is using an ancilla qubit, the statevector is the tensor product between the first n qubits and the ancilla. Thus the actual state you get with the statevector (for the target state $|\beta\rangle$) is:

$$|\beta\rangle \otimes |-\rangle = \frac{1}{\sqrt{2}} \left(a\,|\beta\rangle \otimes |0\rangle - b\,|\beta\rangle \otimes |1\rangle\right)$$

Therefore, the success probability is $|a|^2 + |b|^2$. You can use the method `probabilities_dict()` to obtain a dictionary whose keys are the basis states and the values are the corresponding probabilities[3].

References

[1] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. Quantum computing with Qiskit, 2024.

[2] Charles H Bennett and Stephen J Wiesner. Communication via one-and two-particle operators on einstein-podolsky-rosen states. *Physical review letters*, 69(20):2881, 1992.

[3] Charles H Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K Wootters. Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. *Physical review letters*, 70(13):1895, 1993.

[4] Thomas G Draper. Addition on a quantum computer. *arXiv preprint quant-ph/0008033*, 2000.

[5] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 11–20, 1993.

[6] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[3] *Hint: the bit corresponding to the ancilla qubit is the mostleft one, i.e., that of index 0.*