Prof. M. Chiani, Prof. D. Maio — University of Bologna, Cesena
Tutor: L. Valentini — A.A. 2024/2025

Qiskit Basics

1. *Tutorial: Analysis of a quantum circuit*

- Preliminaries

  (a) Open an empty python file inside the provided folder.

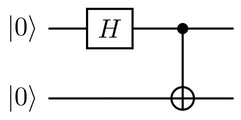
  (b) To verify the correct installation of Qiskit and retrieve its version, use the following command:

```python
import qiskit
print("The current qiskit version is:")
print(qiskit.__version__)
```

  (c) Similarly, you can use the utility we have added inside the Helper Python file.

```python
import HelpersLite as hp
hp.get_qiskit_version()
```

  (d) Construct the quantum circuit:



  by writing the operations in the circuit from left to right (the initial state is already initialized by qiskit to $|00\rangle$):

```python
from qiskit import QuantumCircuit
circ = QuantumCircuit(2, 2)
circ.h(0)
circ.cx(0, 1)
```

  In the code above, `circ = QuantumCircuit`(a, b) create a quantum circuit, named `circ`, with a qubits and b bits.[1] To an instance of a `QuantumCircuit` we can append the quantum gates using the methods (functions) of the class, such as `h`$(\cdot)$, `x`$(\cdot)$, `z`$(\cdot)$, `y`$(\cdot)$, `cx`$(\cdot, \cdot)$. For more gates and details, see the online documentation.

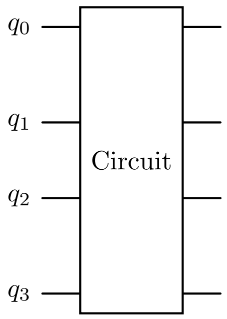
[1]Note that `QuantumCircuit` can also include additional parameters, such as the global phase.

(e) You can visualize your quantum circuit using the `print(·)` function:

```
print("The described circuit is:")
print(circ)
```

- Understanding LSB and MSB in qiskit

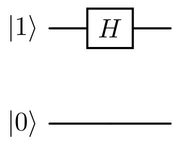
(a) Note that, for describing the initial quantum state $|q_3q_2q_1q_0\rangle$ the qiskit adopts the following notation for the qubit order



Moreover, as commonly assumed in the literature, the state vectors are indexed in ascending order.

Bra-ket Notation	Vector notation	Qiskit notation
$\alpha\|00\rangle + \beta\|10\rangle$	$\begin{pmatrix}\alpha\\0\\\beta\\0\end{pmatrix}$	$[\alpha, 0, \beta, 0]$

(b) *Example:* Consider the following circuit



Then, the input state is $|01\rangle = \begin{pmatrix}0 & 1 & 0 & 0\end{pmatrix}^T = [0,1,0,0]$, while the output state is $|0-\rangle = \frac{|00\rangle - |01\rangle}{\sqrt{2}} = \begin{pmatrix}\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0\end{pmatrix}^T = \left[\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, 0\right]$

- Analysis of a quantum circuit

  (a) To compute the output of a quantum circuit given an input state we use the `Statevector` class:

  ```
  from qiskit.quantum_info import Statevector
  # An array [0, 0, 0, 1] represents the state |11>
  arrayVecIn = np.zeros(4)
  arrayVecIn[3] = 1

  stateVecIn = Statevector(arrayVecIn)
  stateVec = stateVecIn.evolve(circ)
  ```

  The `evolve` method compute the output state vector (`stateVec`), starting from the state vector instance that calls it (`stateVecIn`) and considering the quantum circuit given as an input (`circ`).

  (b) To visualize the results

  ```
  # Print the state vectors
  print("\nInput state vector:")
  print(stateVecIn)
  print("\nOutput state vector:")
  print(stateVec)

  # You can use our utility for plots
  hp.plot_statevector(stateVec, "State Vector")
  ```

  (c) For more complex systems, you can use the `from_label` static methods of `Statevector`. For example, to initialize a two-qubit state to $|+1\rangle = |+\rangle \otimes |1\rangle = (|01\rangle + |11\rangle)/\sqrt{2}$:

  ```
  psi = Statevector.from_label("+1")
  # Probabilities for measuring all qubits
  probs = psi.probabilities()
  print("probabilities: {}".format(probs))
  ```

  Try this state as an input of the quantum circuit described above.

  (d) You can also extract the unitary matrix representing your quantum circuit:

  ```
  from qiskit.quantum_info import Operator
  UnitaryRepresentation = Operator(circ)
  print("\nUnitary matrix representing the circuit:")
  print(UnitaryRepresentation.data)
  ```

2. *Exercise:* Check the behaviour of the following gates extracting the unitary representation of the circuits and evolve the quantum states reported in the table below.

   (a) Single gate:

      i. Pauli operators: **X** and **Y**.

      ii. Hadamard gate **H**.

      iii. Rotations: $\mathbf{R}_z(\theta)$ with $\theta = \pi/2$.

   (b) Double gate:

      i. Controlled-NOT (CNOT): **CX**.

      ii. Controlled-Z (CZ): **CZ**.

      iii. Swap: **SWAP**.

Gate	Matrix Representation	Input	Output
$\mathbf{X}$		$\|0\rangle$	
$\mathbf{Y}$		$\frac{\|0\rangle + j\|1\rangle}{\sqrt{2}}$	
$\mathbf{H}$		$\|0\rangle$	
$\mathbf{R}_z(\pi/2)$		$\frac{\|0\rangle + \|1\rangle}{\sqrt{2}}$	
$\mathbf{CX}_{0,1}$		$\frac{\|00\rangle + j\|10\rangle + \|11\rangle}{\sqrt{3}}$	
$\mathbf{CX}_{1,0}$		$\frac{\|00\rangle + j\|10\rangle + \|11\rangle}{\sqrt{3}}$	
$\mathbf{CZ}$		$\|+-\rangle$	
$\mathbf{SWAP}$		$\frac{\|00\rangle + j\|10\rangle + \|11\rangle}{\sqrt{3}}$	

Notation: $j = \sqrt{-1}$. In $\mathbf{CX}_{c,t}$ c is the control qubit and t is the target qubit.

3. *Tutorial: Ideal simulation of a quantum circuit*

(a) Write the program with the same circuit used in the previous tutorial section:

```python
from qiskit import QuantumCircuit

# Circuit definition #
circ = QuantumCircuit(2, 2)
circ.h(0)
circ.cx(0, 1)
```

(b) Add the measurements at the end of the circuit, and visualize it with a print:

```python
circ.measure([0, 1], [0, 1])

print("The described circuit is:")
print(circ)
```

(c) Import and instantiate the ideal Aer simulator:

```python
from qiskit_aer import AerSimulator
idealSim = AerSimulator()
```

(d) Run the simulator on the defined quantum circuit `circ`:

```python
result_ideal = idealSim.run(circ, shots=1000).result()
```

To specify how many iteration the simulator should execute user the `shots` option of the `run`$(\cdot)$ method.

(e) Get the results:

```python
print("The results of the measurements are: (the format is '
    state': occurrences)")
print(result_ideal.get_counts(0))
```

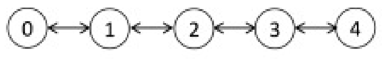
The method `measure`$([q_0, q_1, q_2, ...]$, $[c_0, c_1, c_2, ...])$ store the measurement of the qubit q_i into the classical register c_i. To measure all qubits without specifying the positions, you could use `measure_all(inplace=True, add_bits=False)`

4. *Tutorial: Emulate a quantum computer architecture*

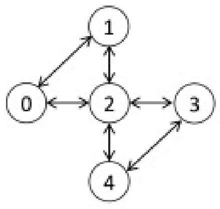
(a) To emulate the 5 qubit quantum computer Athens having a linear connectivity (see figure below), we use the class `CouplingMap` giving as input the list of directed edges (each represented by two qubit indexes) of the qubit architecture graph:

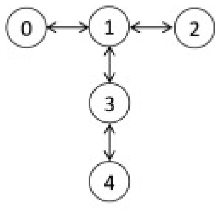
```
from qiskit.transpiler import CouplingMap
architecture = CouplingMap([[0, 1], [1, 2], [2, 3], [3, 4],
    [4, 3], [3, 2], [2, 1], [1, 0]])
```

5 Qubit quantum computer architectures: (*a*) Athens 5 qubit, has the linear connectivity; (*b*) Yorktown 5 qubit, has the bow-tie connectivity; (*c*) Vigo 5 qubit, has the "T" connectivity.



(*a*)





(*b*) (*c*)

(b) List what operations the quantum computer can execute, i.e., the elementary quantum gates the computer can use to construct a generic gate:

```
gatesQcomp = ["id", "rz", "sx", "x", "cx"]
```

(c) Instantiate, with the class `GenericBackendV2`, the custom quantum computer backend:

```
from qiskit.providers.fake_provider import GenericBackendV2
nQubit = 5
customQcomputer = GenericBackendV2(nQubit, gatesQcomp,
    coupling_map=architecture, noise_info=False)
```

(d) Write the following circuit to test:

```python
# Circuit definition #
circ = QuantumCircuit(3, 3)
circ.h(0)
circ.cx(0, 1)
circ.cx(0, 2)
circ.measure([0, 1, 2], [0, 1, 2])

print("The described circuit is:")
print(circ)
```

(e) Transpile the ideal circuit to a quantum circuit executable on the designed custom quantum computer:

```python
from qiskit import transpile
ourSim = AerSimulator.from_backend(customQcomputer)
transpiledCird = transpile(circ, ourSim, optimization_level
    = 0)

print("The transpiled circuit is:")
print(transpiledCirc)
```

Note that the CNOT operation between qubit 0 and 2 in our architecture is not possible and the transpiler has find another way to implement it. Moreover, even with the optimization level set to zero, the transpiler has shuffled the assignment between quantum and classical bits to reduce the overall number of gates.

(f) Transpile with the maximum level of optimization:

```python
# Transpile #
ourSim = AerSimulator.from_backend(customQcomputer)
transpiledCird = transpile(circ, ourSim, optimization_level
    = 3)

print("The transpiled circuit is:")
print(transpiledCirc)
```

Compare the transpiled solution with the previous one.

The output from `transpile`(·) varies due to the stochastic swap mapper. So complex quantum circuits will likely change each time you run the code.