# Qiskit: quantum computing in Python

1

*The use of quantum computers for practical applications, from physics to finance, is becoming a reality.*

*Several parties are building quantum computers and making them available to the public for calculations.*

*Together with the quantum hardware, also the software to run codes on quantum computers is being developed.*

*This article presents an overview of the applications of Qiskit, an open-source framework to run quantum computations in Python.*

Giuseppe Colucci

info.quantumquants@gmail.com

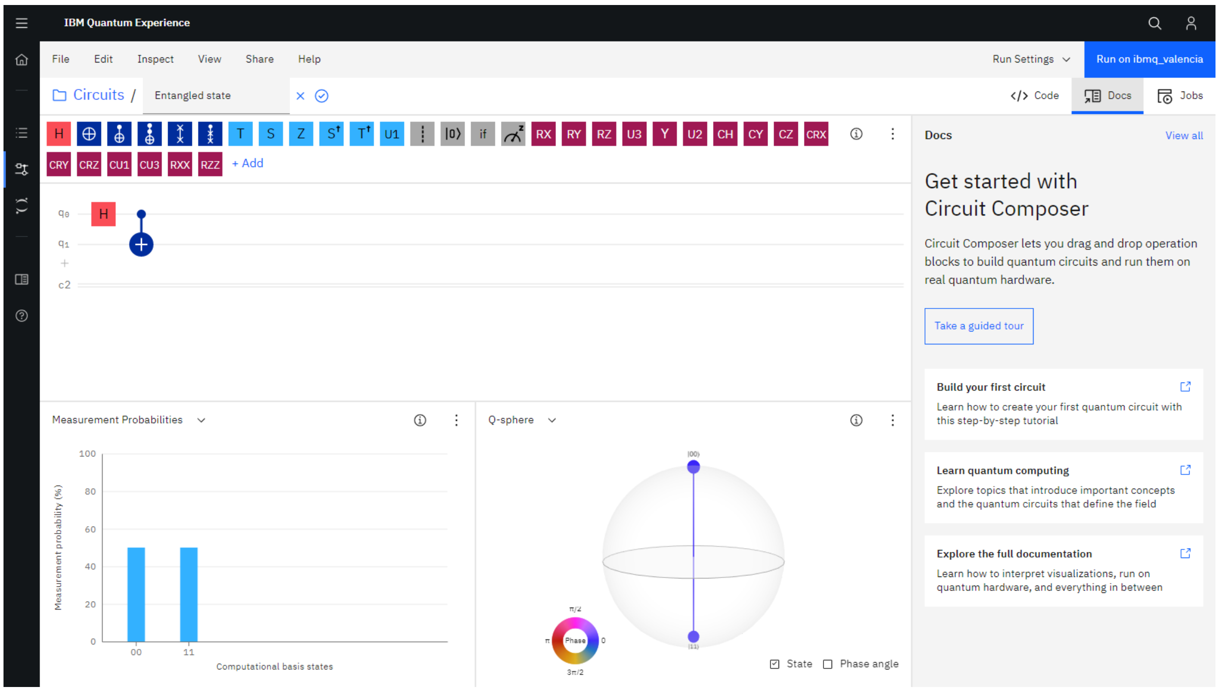
**IBM Q Experience**

The IBM Q Experience[1] is an online platform to explore the possibilities of quantum computing with simulators and real devices (prototypes) developed by IBM.

Once logged in, the interface is very intuitive. The "Quantum Lab" includes a graphical way to build circuits, the Quantum Composer, by drag-and-dropping several elements (quantum gates) which can be applied to the qubits. The obtained circuits can be run both on a simulator and on a real IBM quantum computer.

In Figure 1, an example of a simple quantum circuit is given, which is used to create an entangled state starting from two non-entangled qubits.

[1] https://quantum-computing.ibm.com/



Docs

View all

Get started with Circuit Composer

Circuit Composer lets you drag and drop operation blocks to build quantum circuits and run them on real quantum hardware.

Take a guided tour

**Build your first circuit**
Learn how to create your first quantum circuit with this step-by-step tutorial

**Learn quantum computing**
Explore topics that introduce important concepts and the quantum circuits that define the field

**Explore the full documentation**
Learn how to interpret visualizations, run on quantum hardware, and everything in between

However, the drag-and-drop approach is not the most effective one when the circuits become more and more complex and should be used for practical applications. For this reason, IBM introduced an open-source software development kit (SDK) for quantum computing, Qiskit[2] (pronounced "kiss-kit"), to embed the quantum computing operations in Python.

## Qiskit

Qiskit can be installed locally as a common Python package (e.g. with pip or conda). At the time of writing, python 3.5 or higher is required.

Once installed, Qiskit allows building quantum circuits, running them locally on a simulator or even testing your own quantum circuits on a real IBM quantum computer. For the latter, you must be registered on IBM Q Experience and need to create an API token to be saved locally[3].

The simple circuit in Figure 1 can be run locally from Python by means of the code in Box 1. The code might look involved to run such a simple circuit, however when implementing quantum algorithms involving a large set of gates, the advantages of the SDK become clear.

## Possible applications

Qiskit can be used for both educational, research and practical purposes. The limitation is given by the qubits available on the IBM Q Experience, but research for building quantum computing circuits for practical applications is already a reality.

Qiskit can help the development of quantum gates and circuits, multipurpose quantum algorithms for

[2] https://qiskit.org/

[3] This API token can be found in the settings: https://quantum-computing.ibm.com/account

practical applications (e.g. finance and data science), quantum hardware testing, etc. We at Quantum Quants are currently investigating the possibilities of implementing machine learning and optimization algorithms.

**Box 1: Prepare an entangled state with Qiskit**

```python
# importing Qiskit methods
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.providers.ibmq import least_busy
from qiskit.tools.monitor import job_monitor
from qiskit.visualization import plot_histogram
# NOTE: to use the real IBM quantum chip, you need to be registered to
# the IBM quantum experience website: https://quantum-computing.ibm.com/
# Once registered, one has to save locally the credentials with the command:
# >>> IBMQ.save_account('YOUR_API_TOKEN')
# This API token can be found in the account settings: https://quantum-
computing.ibm.com/account


if __name__ == "__main__":
    # Create the circuit with 2 qubits
    qc = QuantumCircuit(2)
    # Add a Hadamard gate on the first qubit
    qc.h(0)
    # Add a CNOT between the first and second qubit to entangle them
    qc.cx(0, 1)
    # Measure qubits
    qc.measure_all()

    # Draw the circuit
    qc.draw('mpl')

    # Run the circuit on the local simulator
    backend = Aer.get_backend('qasm_simulator')
    shots = 1024
    results = execute(qc, backend=backend, shots=shots).result()
    counts = results.get_counts()
    plot_histogram(counts)

    # Run on a real quantum computer
    # Load the saved IBMQ accounts
    IBMQ.load_account()
    provider = IBMQ.get_provider(hub='ibm-q')
    # Get the least busy backend device with less than or equal to 2
    backend = least_busy(provider.backends(filters=lambda x:
                                            x.configuration().n_qubits >= 2
                                           and not x.configuration().simulator
                                           and x.status().operational == True))

    print("least busy backend: ", backend)

    shots = 2048
    job = execute(qc, backend=backend, shots=shots, optimization_level=3)
    job_monitor(job)

    counts = job.result().get_counts()
    plot_histogram(counts)
```

This code starts from two qubits in the $|0\rangle$ state and transforms them to the maximally entangled state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. The drawing of the circuit is shown in Figure 2.

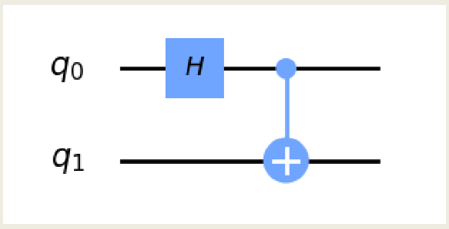


Figure 2: Quantum circuit for preparing a maximally entangled state.

When running on the simulator (Figure 3), the code outputs the state as predicted, with a high accuracy: the probabilities (or square amplitudes) of the states $|00\rangle$ and $|11\rangle$ are not exactly ½ due to the finite number of shots. The result should converge to ½ in the limit of large numbers.

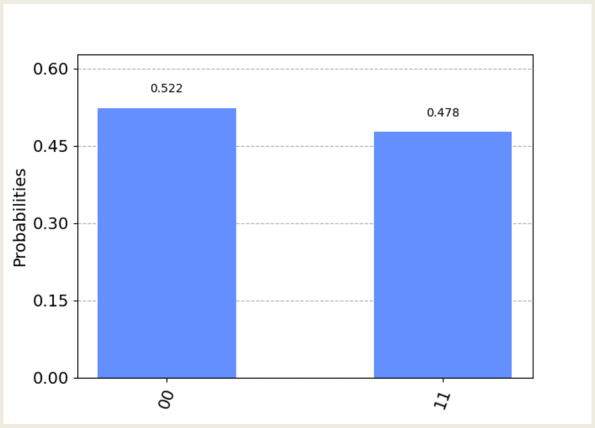


Figure 3: Local simulator results.

Finally, Figure 4 shows the results after running the circuit on an actual quantum device (for this simulation, the least busy backend used was *ibmq_vigo*). The results show a small however non-zero probability for the states $|01\rangle$ and $|10\rangle$. This is the consequence of decoherence and noise in a real quantum computer, which is one of the main difficulties which is currently being addressed to build stable quantum computing devices.

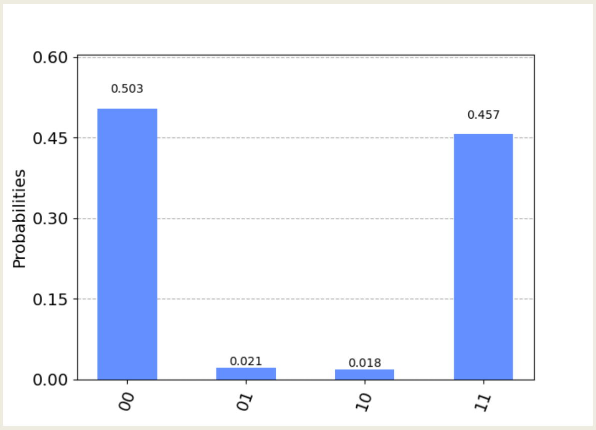


Figure 4: Real device output.

Quiskit already offers several tutorials and an interactive environment to learn the basics and explore the possibilities and applications to several fields (e.g. chemistry, finance, optimization, etc.). Tutorials are provided in the jupyter notebook format[4] and the user can also create and store his/her own notebooks. The Qiskit community is also very active and everyone can contribute to it via Github[5].

Our goal at Quantum quants is to provide the knowledge and necessary means to develop your own Qiskit suite of codes and notebooks for researching the possible applications of quantum computing to your field.

## Conclusions

The Qiskit framework allows for an easy implementation of algorithms that can be run on a quantum computer. The embedding in python allows for using many libraries on the side of the quantum computation. Qiskit can then be used to research, create and implement hybrid algorithm which use the versatility of python libraries like pandas or keras with the calculation on a simulator or a real quantum device.

At Quantum Quants we implemented and built several algorithms, from historically relevant ones like the Simon's and Shor's algorithms, to more practical solutions to calculate correlations of stochastic processes or to build a quantum layer in a neural network architecture.

Currently, we also investigate algorithms and implementations for the calculation of molecules and compounds on a quantum computer. However this requires additional quantum computational power which cannot be achieved with the limited number of qubits provided on the IBM Q Experience platform. Therefore we look for alternatives, including the Google framework of Tensorflow-quantum[6], the AWS Amazon Braket[7] service and the Leap2 framework of D-Wave[8].

[4] https://quantum-computing.ibm.com/jupyter
[5] https://github.com/Qiskit/
[6] https://www.tensorflow.org/quantum
[7] https://aws.amazon.com/braket/
[8] https://www.dwavesys.com/take-leap

## About Quantum Quants

Quantum Quants helps companies to get insights on applications of quantum computing to the financial industry and data science.

Giuseppe Colucci (founder) is a PhD in theoretical physics and ALM specialist at de Volksbank N.V. He is an expert on thermal quantum field theory, risk models and ALM strategy. He developed interest for quantum computing since his master degree in Theoretical Physics and is currently active in publishing academic papers on applications of quantum computing to finance and data science.

Quantum Quants

Rotterdam (The Netherlands)

email: info.quantumquants@gmail.com