Quantum Computation 101 for Physicists

Home exercise - programming 1

1 Simulations

In this exercise you will work for the first time with the QISKIT package of IBM. The QISKIT package was developed to handle quantum computers in the form of circuits. We will work with the simulator of a quantum circuit on a classical computer, but you can use the exact same package if you ever want to perform an experiment on IBM's quantum computer, which is composed of 50 qubits as for today.

1.1 Installation

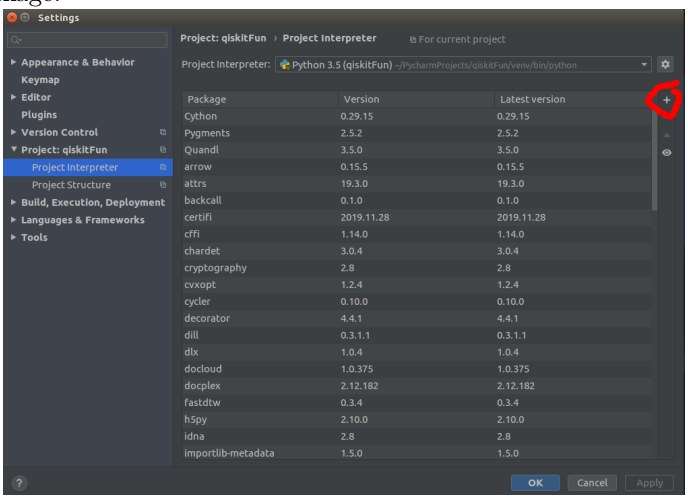
The programming language QISKIT is costumed for is Python. If you do not have Python installed on your computer, download Python 3 from the following link:

https://www.python.org/downloads/.

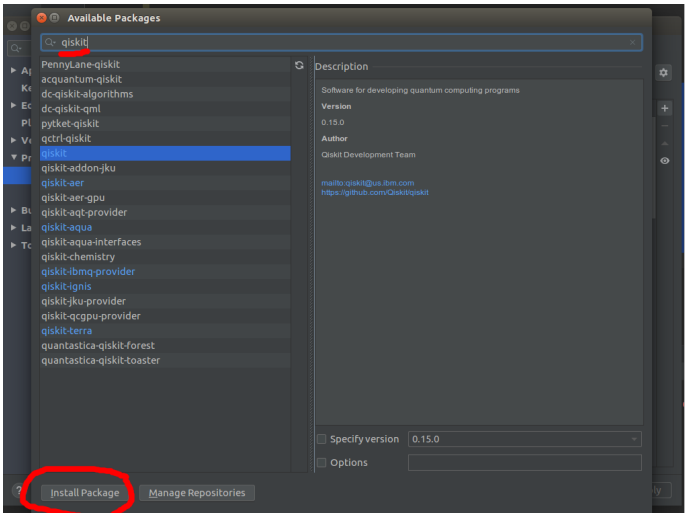
It is recomended to install PyCharm as a platform for writing Python code:

https://www.jetbrains.com/help/pycharm/installation-guide.html.

Open a PyCharm project using **File>New Project**. Open the project interpreter settings using **File** > **Settings** > **Project** > **Project Interpreter**. Hit the '+' botton to install a new package.



In the search line, write 'qiskit'. After the package was found, press 'Install package'.



In the same way, you will need to instal the **matplotlib** package.

Note: You may need to install pip for the package installation to work. Use the following link for the installation: https://www.liquidweb.com/kb/install-pip-windows/

1.2 Basic operations

We start by adding the following lines. Aer is the classical simulator for quantum circuits:

```
from qiskit import(
execute,
Aer)

# Use Aer's qasm_simulator
simulator = Aer.get_backend('qasm_simulator')

```

We create a qircuit using the QuantumCircuit component:

```
from qiskit import QuantumCircuit
circuit = QuantumCircuit(qbits, cbits)

```

Here *qbits* is the number of qubits in the circuit, and *cbits* in the number of classical bits in which we will store measurement results.

After creating a circuit, we can apply quantum gates to it in the following way:

```
circuit.h(3)

```

Here we applied the Hadamard gate to the qubit with index 3 in the circuit. Important gates you will need to know for this exercise are:

```
# Hadamard gate
circuit.h(qubit_number)

# X gate
circuit.x(qubit_number)

# Z gate
circuit.z(qubit_number)

# CNOT gate with control qubit control_qubit and target qubit target_qubit
circuit.cx(control_qubit, target_qubit)

```

We can add a measurement of one or several qubits into the classical bits, using:

```
# Measure the qubits in qibits_subset and store the result in the classical bits
  cbits_subset. Note that we do not have to measure all of the qubits, but only
  the subset we choose.
circuit.measure(qubits_subset, cbits_subset)

```

After we sequentially add quantum gates and measures, we can draw the circuit into a figure:

```
# Draw the circuit
circuit.draw(output='mpl', filename='my_circuit.png')

```

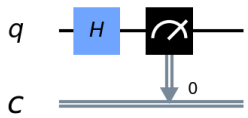
We can now run a simulation of this circuit, using the *simulator* we initiated earlier:

```
# Execute the circuit on the IBM simulator
# The simulator will run 1000 simulations of the circuit and will collect the measurement results.
job = execute(circuit, simulator, shots=1000)

# Grab results from the job
result = job.result()

# Returns counts of the number of occurences of each eigenstate of the computational basis
counts = result.get_counts(circuit)
print(counts)

```

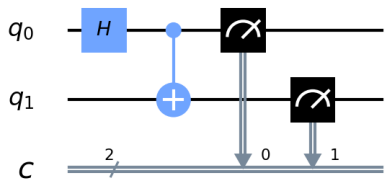
A usefull link you can use to learn more about quantum gates is https://qiskit.org/textbook/ch-gates/quantum-gates.html.

1.3 Questions

1. Create the following gates. Run a simulation with 1000 shots and write down the counts for each computational basis eigenstate of the system:



(a)



(b)

2. Create the state $\frac{1}{2^3}\sum_{i=1}^{2^6}|i\rangle_6$ using the result from class. Run a simulation with $2^6 \cdot 1000$ shots and get the count for the eigenstate $|31\rangle_6$. Compare with the expectation value of $|31\rangle_6$.

3. Create a circuit that acts the following way on a state of zeros: the state $|00000\rangle \rightarrow \frac{1}{\sqrt{2}}(|00000\rangle + |11111\rangle)$. Draw the circle using QISKIT and write the output counts for a 1000 shots simulation.

4. Write a program in QISKIT that implements the quantum teleportation procedure you saw in class (which you can find in chapter 6.5 in Mermin's book).

   (a) Start from a 3-qubit circuit. Create 4 different registers: 1 classical register and one quantum register for Alice and 1 classical register and one quantum register for Bob:

```

        aliceQubits = QuantumRegister(2, 'a')
        aliceCBits = ClassicalRegister(2, 'ac')
        bobQubits = QuantumRegister(1, 'b')
        bobCBits = ClassicalRegister(1, 'bc')

        circuit = QuantumCircuit(aliceQubits, bobQubits, aliceCBits, bobCBits)

```

(b) Rotate the first qubit around the x axis by some angles θ and ϕ of your choice to get the first qubit in the state $|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\phi}\sin(\theta/2)|1\rangle$. The gate used for rotation can be found here: https://qiskit.org/textbook/ch-states/single-qubit-gates.html#7.-The-U-gate–. Note that this is not the most general state for a single qubit, as we saw last week.

```
        circuit.u(theta, phi, 0, aliceQubits[0])

```

(c) Get qubits 2 and 3 to a Bell pair $|B_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

(d) Follow the procedure in page 150 in Mermin's book in order to 'teleport' the state of the first qubit into the third qubit. Note, as Alice has qubits 1 and 2, and Bob has qubit 3, no gates can be applied on qubits (1, 3) or (2, 3). You will need to use a new method of QISKIT called c_if, which allows you to apply gates based on measurements results:

```
        # Measure qubits 0 and 1 and based on the measurement, apply gates
        circuit.measure(aliceQubits, aliceCbits)
        # If the result is 10 = 2
        circuit.applySomeGate(bobQubits[0]).c_if(aliceCBits, 2)

```

Here we first convert the result n the classical bits to the binary number they represent, (2 in the example), and add it as the second argument to the c_if method.

(e) Measure qubit 3 in the computational basis and run 1000 shots of the circuit. Compare with your expected result from (b).

(f) Draw the circuit. Name your code file **teleportation.py** and submit your code along with the drawing of your circuit.