CSCI 4140 Laboratory Two

Working with Qubits

Introduction

The purpose of this laboratory is to become familiar with qubits and gates. By the end of this lab you will be able to construct quantum circuits, simulate their operation and visualize the results. At several points in this laboratory you will be asked to cut and paste your results into a laboratory report. You can use whatever software you like to construct your report, but it must be submitted as a pdf file. You should work on the report as you work through the lab. The easiest way to do this lab is using Jupyter notebook.

Single Qubit Circuits

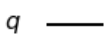
Start the lab with a few simple single qubit circuits. Start a Jupyter notebook for this lab and in the first cell put the import statements that you need:

```
In [1]: from qiskit import QuantumCircuit, execute, Aer
        from qiskit.visualization import plot_histogram, plot_bloch_multivector
        from math import sqrt, pi
```

Start by creating a single qubit and draw the circuit like we've seen in the lecture.

```
In [2]: qc=QuantumCircuit(1)
        qc.draw('mpl')
```

Out[2]:



Now add an X gate before the draw call and run the cell. Now we will simulate the circuit and view the result. Again, this is similar to the video lecture. First comment out the draw call, since we can only have one visualization. Next retrieve the state_vector simulator, execute the circuit and view the result in a histogram. Your cell should look like the following:

```
: qc = QuantumCircuit(1)
  qc.x(0)
  #qc.draw('mpl')
  backend = Aer.get_backend('statevector_simulator')
  result = execute(qc, backend).result()
  print(result.get_statevector())
  plot_histogram(result.get_counts())
```

Now replace the plot_histogram call with a call to plot_bloch_multivector. Remember to use the state vector and not the counts as the parameter to this procedure. Cut and paste your code and the resulting Bloch sphere into your laboratory report.

In the video lecture we examined the rz gate, we can do the same thing around the Y axis using the ry gate. After the x gate in our example add qc.ry(pi/2.0, 0). Observe the impact that this has on the resulting state vector. Try a few angles and see what the result looks like. Recall from the video lecture that basically the only thing that we can do with a single qubit is rotate it.

Multiple Qubit Circuit

We need more than one qubit to do anything interesting. Most of the gates that we will be interested in are two qubit gates. There are three qubit gates, but they are hard to implement in hardware. Start by building two qubits, with the first qubit in state |0> and the second in state |1>. You should have something like the following:

```
In [9]: qc = QuantumCircuit(2)
        qc.x(1)
        qc.draw('mpl')
```

Out[9]:

In the video lecture we used the unitary simulator to produce the matrix corresponding to this circuit. We will start by repeating that:

```
In [12]: qc = QuantumCircuit(2)
         qc.x(1)
         backend = Aer.get_backend('unitary_simulator')
         unitary = execute(qc, backend).result().get_unitary()
         from qiskit_textbook.tools import array_to_latex
         array_to_latex(unitary, pretext="\\text{Circuit =}\n")
```

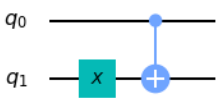
$$\text{Circuit} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Now add a cx(0,1) to the circuit and note the change in the matrix. Now add an h gate on the first qubit (index 0) before the cx gate and observe the matrix. Cut and paste your code and the result into your laboratory report. As you can see the matrices get complicated as we add gates. If we had three qubits, we would be dealing with 8x8 matrices, so as our circuits get larger, the matrices get larger and we eventually run out of memory.

Now let's go back and examine the controlled gates in more detail. Start by examining what they look like in circuit diagrams. Build a circuit with a cx gate and display the circuit:

```
In [15]: qc = QuantumCircuit(2)
         qc.x(1)
         qc.cx(0,1)
         qc.draw('mpl')
```

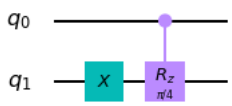
Out[15]:



Note that the small dot is on the line for the control qubit and the "+" is on the line for the controlled qubit. Replace the cx by a cy and a cz and observe how the circuits change. The following show what happens when we add a controlled rz to the circuit:

```
In [18]: qc = QuantumCircuit(2)
         qc.x(1)
         qc.crz(pi/4.0,0,1)
         qc.draw('mpl')
```

Out[18]:



When developing a quantum program, it is always a good idea to display the circuit before you try to simulate it. It's easier to catch bugs at this point, rather than trying to determine what went wrong from the simulation results.

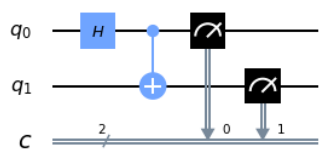
Making it Real

Both the state vector and unitary simulators are perfect mathematical simulations of the circuit. This is good for debugging, since we don't want random things happening when we are debugging a program. Unfortunately, this isn't very realistic, since existing quantum computers have noise which adds some randomness to our results. To evaluate how a circuit will perform on a real computer we need a simulator that will add noise to the computation in a realistic way. The qasm simulator will do this for us.

To see how this works we will return to our entangled state circuit. In order to get reasonable results from our qasm simulator we need to add a measurement to our circuit. This is done with the measure procedure. This procedure takes two parameters. The first parameter is the list of qubits that will be measured, and the second is a list of the classical bits where the result will be placed. We also need to change our QuantumCircuit call to add 2 classical bits to our circuit.

```
In [44]: qc = QuantumCircuit(2,2)
         qc.h(0)
         qc.cx(0,1)
         qc.measure([0,1],[0,1])
         qc.draw('mpl')
```

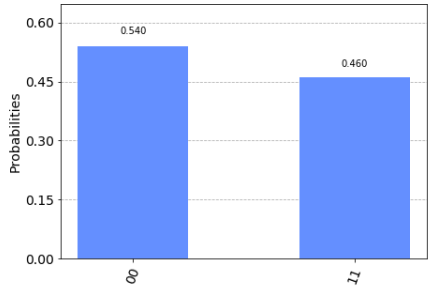
Out[44]:



Now we can add the qasm simulator and view the results. Note that in this case we aren't getting the perfect 0.5 probabilities we get in the other simulators. Each time you run this program you will get different results.

In [45]:
```
qc = QuantumCircuit(2,2)
qc.h(0)
qc.cx(0,1)
qc.measure([0,1],[0,1])
backend=Aer.get_backend('qasm_simulator')
result = execute(qc, backend, shots=100).result()
plot_histogram(result.get_counts())
```

Out[45]:



The shots parameter to the execute procedure is the number of times that the simulation is run. Increase the value of shots to 1,000, 10,000 and 100,000 and see the results. By 100,000 runs you should be getting close to the theoretical value. Cut and paste your results for the 100,000 run and add them to your report.

Laboratory Report

Convert your report to a pdf file and submit it through Canvas.