CSCI 4140 Assignment One

Due October 23, 2020

Introduction

In laboratory 4 we investigated the quantum phase estimation algorithm and you implemented it using perfect qubits. In this laboratory we will examine this algorithm in the real world where we need to deal with noise and a limited number of gates. This assignment involves a number of steps that you can follow to produce the final solution. Recall that we had two implementations of the algorithm. First, the exact version where we could get the exact phase. This was called qpe in our program. Second, the approximate version where we could only achieve an approximation of the phase. This was called qpe2 or qpe3 in our program. We will work with both of these versions.

Generalizing the Algorithm

Both the exact and approximate implementations of the algorithm worked with a fixed number of qubits. In this part of the assignment modify both programs to have a parameter, n, which is the number of qubits. Note, the number of classical bits will also depend on n. This will allow us to easily run experiments with different numbers of qubits. Check your new version of the code against the previous implementations with a fixed number of qubits to make sure that they are producing the same circuit when n=3.

You can start off with the following set of import statements:

```
import matplotlib.pyplot as plt
import numpy as np
import math
from numpy import pi

# importing Qiskit
from qiskit import IBMQ, Aer
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, execute

# import basic plot tools
from qiskit.visualization import plot_histogram
from qiskit.providers.aer.noise import NoiseModel
from qiskit.providers.aer.noise.errors import pauli_error, depolarizing_error
```

Adding Noise

So far we have been dealing with perfect qubits, but this isn't the case in the real world. Existing quantum computers have noise, which impact gates and measurements. Our first question is how seriously does noise effect our results? In order to do this we need a noise model, which I provide for you. This noise model is in the following procedure which is similar to one in the Qiskit textbook, modified for the purposes of our experiment. This procedure is:

```
def get_noise(p_meas,p_gate):

    error_meas = pauli_error([('X',p_meas), ('I', 1 - p_meas)])
    error_gate1 = depolarizing_error(p_gate, 1)
    error_gate2 = error_gate1.tensor(error_gate1)

    noise_model = NoiseModel()
    noise_model.add_all_qubit_quantum_error(error_meas, "measure")
    noise_model.add_all_qubit_quantum_error(error_gate1, ["u1", "u2", "u3"])
    noise_model.add_all_qubit_quantum_error(error_gate2, ["cx", "cu1"])
    return noise_model
```

The first parameter to this procedure is the noise level for measurement. This will be left at 0.01 for our experiments. The second parameter is the gate noise level, which will be varied, but will start at 0.01.

This noise model is used in the following way:

```
noise_model = get_noise(0.01,0.01)
backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots,
noise_model=noise_model).result()
answer = results.get_counts()

plot_histogram(answer)
```

Now we are ready for our first experiment. For this experiment we will use two noise levels, 0.01 and 0.05 and three numbers of qubits, 6, 8 and10. We also have the exact and approximate algorithms. This gives us 12 possible combinations to try. Run these experiments and record your results in two tables, one for the exact version of the algorithm and one for the approximate version of the algorithm.

Number of Gates

It is a widely held belief that with a large enough number of qubits some of the gates in the inverse quantum Fourier transform can be ignored. This is based on the observation that as we add more qubits, we add more gates with very small rotations. It has been argued that these rotations are hard to implement on real hardware and contribute little to the final result. Time for an experiment to see if this is actually the case.

The following version of the inverse QFT removes the smaller rotations:

```
def qft_limited(circ, n):
    """n-qubit QFTdagger the first n qubits in circ"""
```

```
# Don't forget the Swaps!
for qubit in range(n//2):
   circ.swap(qubit, n-qubit-1)
for j in range(n):
   for m in range(j):
      if (j-m) < 3:
         circ.cu1(-pi/float(2**(j-m)), m, j)
   circ.h(j)
```

Note, that I've just added a single if statement to the code that eliminates the smaller rotations. In this case I'm comparing j-m to 3, you can try other values, but this is a good starting point. Again, run an experiment with both algorithms and 6, 8 and 10 qubits. You can use a single noise level for these experiments. Record your results in two tables just like you did with the previous experiment. Do your results show that this is a good assumption?

Assignment Report

Your report for this assignment must include your code for the two versions of the algorithms, plus the four tables that your produced. I don't need to see all of the code, just the two algorithms. Add a paragraph or two discussing your conclusions for these experiments. Submit the report as a PDF file using Canvas.