SKKU 2024 Challenge - Quantum State Tomography

Jason Iaconis

Introduction

In quantum computing, we process information by applying quantum circuits to create entangled *quantum states*. At the end of the computation, we need to extract classical information from these quantum states in order to use the results of our computation. This extraction of information is accomplished by performing *measurements* on the prepared quantum state. A general quantum state on *n* qubits can be written as a superposition over 2^n basis states, each of which has an amplitude which is represented by a *complex number.*

Usually, we design quantum algorithms so that we can extract the relevant information from a quantum state without needing to measure every amplitude of the full final wave function. However, in certain cases it is desirable to reconstruct the (unknown) output state of a quantum circuit. The most common use case for this is to verify and validate the performance of quantum hardware.

In this challenge, we are going to better understand how one can reconstruct unknown quantum states from measurements performed on a quantum computer. In order to reconstruct both the amplitude and phase information of a general quantum state, we need to perform measurements in different bases. The way we measure in different bases is to apply a sequence of quantum gates to the unknown state, and then measure in the computational basis. Throughout this challenge, we will be given an "unknown" quantum circuit, U_{hidden}, that creates our hidden state. We can then append up to 20 different circuit at the end of U_{hidden}, and measure the resulting state in the computational basis with 100 shots each. Our goal is to create a new quantum circuit, U_{gen}, using only the information gained from these measurements, such that $|<0|\,{U_{gen}}^{\dagger}U_{hidden}|0>|\,\approx 1$.

Problem 1: Quantum States with Few Qubits (☆)

The first class of states that we will consider are generic pure quantum states (i.e. there is no restriction on the quantum circuit which created the state), on a small number of qubits. In general, a generic quantum state will require an exponential number of measurements to reconstruct. However, when the number of qubits is small, we can simply pay this exponential cost.

Part 1 (Easy):

In this first problem, we have a single qubit quantum state

$$|\psi\rangle = a|0\rangle + b\, e^{i\theta}\, |1\rangle,$$

where we can treat the unknown parameters a, b and θ as real positive numbers with $a^2 + b^2 = 1$ and $\theta \in [0, 2\pi]$.

a) Explain how one can determine the values of a, b and θ by measuring the quantum state in different bases.

Hints: First determine the values of the amplitudes a and b . In order to determine the phase θ, you will need to generate interference between the amplitude of the two basis states. Consider the effect on the quantum state of applying combinations of the simplest single qubit gates (the Hadamard and S gates).

b) Write a program in Qiskit which generates this unknown quantum state.

To generate a random single qubit quantum state we use a circuit U_{hidden}, so that $|\psi\rangle = U_{hidden}|0\rangle$. An example hidden state can be created with the code:

```
import numpy as np
from qiskit import *

def create_1q_target_circuit(qc_data):

    qc = QuantumCircuit( 1)
    qc.ry(qc_data[ 0], 0)
    qc.rz(qc_data[ 1], 0)

    return qc

random_vec = np.random.rand( 2)* 2. 0*np.pi
U_hidden = create_1q_target_circuit(random_vec)
```

Important: This code is only for testing your measurement protocol. The rest of your program should proceed as if you do not know the quantum circuit used to create U_{hidden}.

You can measure in different bases by appending your quantum circuit at the end of U_{hidden} .

Measurements at the end of this circuit should use no more than 100 shots each. You may measure in up to 20 different bases. (Note: you will not need to use this many measurement bases for this simple case).

You can perform this measurement by adapting the code

```
backend = Aer.get_backend( 'aer_simulator' )
qc1 = U_hidden.copy()
qc2 = QuantumCircuit( 1)

>> add gates to qc2

circ = qc1.compose(qc2)
circ.measure_all()
result = backend.run(circ,shots= 100).result()
```

After you determine the amplitudes of the hidden state as well as possible, create a quantum circuit which generates the hidden state with as high an overlap as possible (Hint: look into Qiskit's "prepare_state" function).

If U_{gen} effectively reconstructs the hidden state, then applying the inverse circuit to the hidden state should result in a wavefunction which is close to the starting state.

That is, you should find

$$U_{gen}^{\dagger} U_{hidden}|0\rangle \approx |0\rangle$$

so that most of the amplitude is on the all zero basis state.

You can measure how well U_{gen} reconstructs the state using the function below.

```
backend2 = Aer.get_backend('statevector_simulator')
def give_score(U_hidden, U_gen):

    qc = U_hidden.compose(U_gen.inverse())
    result = backend2.run(qc).result()
    score = result.get_statevector()[0]

    return np.abs(score)
```

A score of 1.0 implies a perfect reconstruction of the hidden wavefunction. You should aim to achieve a score > 0.95

Part II (Medium) : We will now increase the complexity by reconstructing a generic two-qubit quantum state.

$$|\psi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle,$$

Where $|a|^2 + |b|^2 + |c|^2 + |d^2| = 1$.

a) First, consider the case where all amplitudes are real numbers. Explain how you can measure both the magnitude and sign of the unknown amplitudes, measuring in as few different bases as possible.

b) Now, explain how you would adapt this measurement protocol if the amplitudes a, b, c, d are complex numbers. Note, that we can only reconstruct the wavefunction up to a global phase, so that we can always consider a to be a positive real number.

c) Now write a program in Qiskit which will generate this unknown quantum state. Use the same format as for part I. You may generate a test random state using the code

```
def create_2q_target_circuit(qc_data):

    qc = QuantumCircuit( 2)
    for i in range ( 2):
        qc.ry(qc_data[ 2*i+ 1],i)
        qc.rz(qc_data[ 2*i+ 1],i)

    qc.cx( 0, 1)
    qc.rz(qc_data[ 4], 1)
    qc.cx( 0, 1)

    for i in range ( 2):
        qc.ry(qc_data[ 4+2*i+ 1],i)
        qc.rz(qc_data[ 4+2*i+ 1],i)

    return qc

random_vec = np.random.rand( 9)* 2. 0*np.pi
U_hidden = create_2q_target_circuit(random_vec)
```

Again, you should develop a series of measurement circuits to append to the end of U_{hidden}. Use the same number of shots and measurement circuits as in Part I.

Then create a quantum circuit U_{gen}, which has as high an overlap as possible with $U_{hidden}|\vec{0}\rangle$.

You should get as high a score as possible using the same scoring function as in part I)

Part III) (Medium)

Describe how you could extend the protocol developed in part's I) and II) to a general *n* qubit quantum state (You do NOT need to code a solution for this part). How many different measurements would you need to apply?

Problem 2 : Graph States (Medium / Hard) (☆)

In some cases, one can reconstruct a quantum state using only a limited number of measurements, regardless of how many qubits are in the quantum system. One of these cases is when the quantum states are generated using only "Clifford gates" (these are essentially quantum circuits that use only H,S, CNOT and CZ gates). In this problem, we will perform quantum state tomography on a subset of Clifford states known as "graph states". A graph state on n qubits is any state which can be generated using a circuit of the form

$$|\psi\rangle = \left(\prod_{(i,j) \in G} CZ_{ij} \right) H^{\otimes n} |\vec{0}\rangle$$

Where CZ_{ij} is a controlled-Z gate applied on qubits i and j, and (i,j) are edges of a hidden graph G. Different graphs, G, create different graph states. Our goal is to determine generate the graph state, $|\psi\rangle$, without knowing the specific graph which was used.

a) Look into the properties of graph states (https://en.wikipedia.org/wiki/Graph_state) . Graph states are stabilizer states with a specific structure. Describe the key properties of stabilizer states. Using the stabilizer properties of graph states, describe how one can perform tomography on graph states.

b)
You may use the following code snippet to generate a random graph state

```
def generate_clifford_circ(self, num_qubits, depth):

    U_hidden = QuantumCircuit(num_qubits)

    for i in range (num_qubit):
        U_hidden.append(qc.h(i))

    count = 0;

    s1s2 = { - 1, - 1}
    while (count<depth):
```

```
        site_i = int(np.random.rand()*num_qubits)
        site_j = int(np.random.rand()*num_qubits)
        if(site_i==site_j or {site_i,site_j}==s1s2 ):
            continue
        s1s2 = {site_i,site_j};

        count+=1
        U_hidden.cz(site_i,site_j)

    return U_hidden
```

□

You may set `num_qubits < 20`.

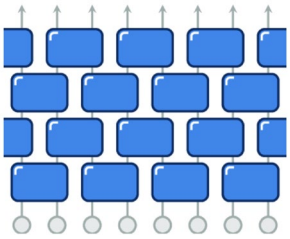
Following the format of **Problem 1**, measure U_{hidden} in `<= 20` bases by appendix quantum circuits to U_{hidden} and taking no more than 100 shots per basis. Then create a new circuit U_{gen} such that

$$U_{gen}{}^{\dagger}\, U_{hidden}|0\rangle \approx |0\rangle.$$

You can again measure the success of your method using the `give_score` function from Problem 1.

Problem 3 : Low Depth Brickwork Circuits (Hard) (☆)

A third class of quantum states for which we can efficiently perform tomography are states which are not very highly entangled. Such states can be generated using a “brickwork” circuit of low depth, with a structure shown below. Each two qubit gate acts only between neighboring qubits.



We consider a quantum state generated by such a brickwork circuit with only two rows of gates (“one brickwork layer”).

This state can be generated using the following code snippet:

```
def low_depth(num_qubits):

    qc = QuantumCircuit(num_qubits)
    for i in range(0,num_qubits-1, 2):

        params = np.random.rand(15)* 2*np.pi
        qc.u(params[0],params[1],params[2],i)
        qc.u(params[3],params[4],params[5],i+1)
        qc.rxx(params[6],i,i+1)
        qc.ryy(params[7],i,i+1)
        qc.rzz(params[8],i,i+1)
        qc.u(params[9],params[10],params[11],i)
        qc.u(params[12],params[13],params[14],i+1)

    for i in range(1,num_qubits-(num_qubits+1)%2, 2):

        params = np.random.rand(15)* 2*np.pi
        qc.u(params[0],params[1],params[2],i)
        qc.u(params[3],params[4],params[5],i+1)
        qc.rxx(params[6],i,i+1)
        qc.ryy(params[7],i,i+1)
        qc.rzz(params[8],i,i+1)
        qc.u(params[9],params[10],params[11],i)
        qc.u(params[12],params[13],params[14],i+1)

    U_hidden = qc
    return U_hidden
```

Based on the information gained by appending ≤ 20 arbitrary quantum circuits to the end of U_{hidden}, and taking 100 shots each, create a quantum circuit U_{gen} such that

$$U_{gen}{}^{\dagger}\, U_{hidden}|0\rangle$$

gives as high of a score as possible using the `give_score` function given in Problem 1.

Be creative and use whatever method gives the highest score possible. Achieving a score >0.5 on most circuits for this problem will be considered good.