Aleksei Tourkine

Introduction

The text below explains the solution for the 2 tasks. The task contains 4 steps :

1) Preparing the Bell state
2) Implementation of arbitrary “error gate”
3) Encoding the error correction
4) Testing the solution and discussion of the results.

Part I. The Bell state

The code for the Bell state is in the **bellstate.py**:

```
qb1 = qskt.QuantumRegister(dimension, 'qb1')
qb2 = qskt.QuantumRegister(dimension, 'qb2')
cm = qskt.ClassicalRegister(2)

circuit = qskt.QuantumCircuit(qb1, qb2, cm)

# Basic circuit
circuit.h(qb1)
circuit.cx(qb1, qb2)

circuit.measure(qb1, cm[0])
circuit.measure(qb2, cm[1])
```

Part II. Simulating quantum noise

In order to simulate the noise two solutions are possible: using the built-in qiskit noise generator or using additional qubits to add the noise. The second solution requires two qubits of noise per one underlying qubit. One additional qubit is used to generate the error event and the other is to choose the error type: bit flip or sign flip. Below is the code from **BellStateWithNoise.py**:

```
def errorCode(input, errorQb, errorTypeQb, errorProba, bitFlipProba, circuit):
    # circuit.
    for i in range(len(input)):
        circuit.initialize([sqrt(1 - errorProba), sqrt(errorProba)], errorQb[i])
        circuit.initialize([sqrt(1 - bitFlipProba), sqrt(bitFlipProba)], errorTypeQb[i])
        circuit.ccx(errorQb[i], errorTypeQb[i], input[i])

        circuit.x(errorTypeQb[i])
        ccz(errorQb[i], errorTypeQb[i], input[i], circuit)
```

Consider a void circuit with 30 pct probability of error and 70 probability of bit flip in case of error. Running the file with these parameters gives (for 10000 simulations)
{'00': 3947, '01': 1019, '10': 1062, '11': 3972}
As expected the noise affects 21% of scenarios. This corresponds to the probability of the bit-flip error. The sign-flip error cannot be observed in this way. To do that one can uncompute the circuit and measure the first qubit only

```
circuit.cx(qb1, qb2)
circuit.h(qb1)
circuit.measure(qb1, cm[0])
```

which yields the results: {'00': 9052, '01': 948} in accorance with the expected probability:

9% = 30% * (1-70%)

If we observe both qubits after the full uncomputation of the circuit

{'00': 7175, '01': 708, '10': 1934, '11': 183}

total 28% of error affected states is observed close to 30% expected.

The alternative solution is in the **NoiseModelTask2.py**.

```
def buildNoiseModel(errorProba, probaType):
   noise_bit_flip = NoiseModel()
   error = pauli_error([('X', probaType * errorProba), ('I', 1 - errorProba), ('Z', (1 -
probaType) * errorProba)])
   noise_bit_flip.add_all_qubit_quantum_error(error, ["id"])
   return noise_bit_flip
```

With the noise model above the noise will be generated by the **"id"** gates:

```
circ.id(qb)
```

The advantage of using the built-in noise generation is that it economizes the qubits and thus reduces the simulation time.

Part III. Error correction

Single qubit correction

First I am going to briefly discuss the one-dimension case following ideas from the wikipedia article referenced in the task description. Please see the file **BasicErrorCorrectionCircuit,py**

Let’s first simulate the qubit repeated 3 times.

{'000': 4979, '001': 1324, '010': 1247, '011': 354, '100': 1276, '101': 374, '110': 351, '111': 95}

To define the correct value we can measure only the last two bits and establish the rule as:

- if the last two bits differ then take the first bit as is
- otherwise if both are equal the result is equal to their value

In that case we will be mistaken about the first bit only in cases

’011': 354, ‘101': 374, '110': 351, '111': 95

The total probability being 11.7% and we will correctly guess the value of the first bit in 88.3% which is already better than the initial situation.

Now we can make the error correction automatic with the following code:

```
def bitFlipCode(input, ancilla, circuit):
# bit flip correction ------------------------------------------------
   circuit.cx(input[0], ancilla[0])
   circuit.cx(input[1], ancilla[0])
   circuit.cx(input[0], ancilla[1])
   circuit.cx(input[2], ancilla[1])

   circuit.ccx(ancilla[0], ancilla[1], input[0])

errorCode(qb1, errorQB[:3], errorQB[3:6], 0.3, 0.7, circuit)
bitFlipCode(qb1, ancilla, circuit)
circuit.measure(qb1[0], cm[0])
```

The principle is the same but there is no need to analyze the measurements. On the other hand we need to use two additional ancilla bits. The quality of the error correction is clearly the same. With the code below

```
# Error code
errorCode(qb1, errorQB[:dimension], errorQB[dimension : (2*dimension)], 0.3, 0.7,
circuit)
# bit-flip correction code
bitFlipCode(qb1, ancilla, circuit)
# measuring
circuit.measure(qb1[0], cm[0])
```

the results are in line with the manual calculation presented above.

```
{'000': 8859, '001': 1141}
```

To fix the sign-flip error one needs to rotate the basis and perform the same procedure:

```
def signFlipCode(input, ancilla, circuit):
   circuit.h(input)
   bitFlipCode(input, ancilla, circuit)
   circuit.h(input)
```

To verify that it works one needs to run the circuit on the balanced state as in the code below.

```
circuit.h(qb1)
# Error code
errorCode(qb1, errorQB[:dimension], errorQB[dimension : (2*dimension)], 0.3, 0.7,
circuit)
signFlipCode(qb1, ancilla, circuit)
circuit.h(qb1)
# measuring
circuit.measure(qb1[0], cm[0])
```

It is important to obeserve that in case of the balanced state the error rate was equal to the probabilty of the sign-flip while for a simple state only the bit-flip error afftcted the result.

Noise correction for the Bell states

Now let's get back to the initial circuit. Let's first anayze the errors affecting the circuit. To that I propose to uncompute the circuit. As in the code below (see **BellStateWithNoise.py**)

```
circuit.h(qb1)

# Error code
errorCode(qb1, errorQB[:dimension], errorQB[dimension : (2*dimension)], 0.3, 0.7, circuit)
errorCode(qb2, errorQB[(2*dimension):(3*dimension)], errorQB[(3*dimension):], 0.3, 0.7,
circuit)

circuit.cx(qb1, qb2)

# Uncomputing
circuit.cx(qb1, qb2)
circuit.h(qb1)

circuit.measure(qb1, cm[0])
circuit.measure(qb2, cm[1])
```

In case there is no error the result will be 00 in all cases, otherwise we'll see a proportion of 1 that corresponds to the error probabilities.

Important observation: we can verify the bit flip error has no effect on the first bit as well as the sign bit error has no effect on the second one. This is an important observation that well simplify the error correction.

It might be tempting to use the error correction described earlier in a straightforward way but it does not actually work. Entanglement of the second qubit with a balanced state complicates things as error-induced uncertainty is mixed with the rotation induced one.

To avoid that we either need to adapt the error correction idea (**Solution 1**) to the current situation or entangle the duplicated qubits before the Hadamard gate (**Solution 2**) .

Solution 1

Let's explore the first approach. Suppose there is only a bit flip error that is happening on the second qubit, In that case we get the following state after the error is introduced:

CNOT [(A* |+0> + B * |+1>)] = A * (|00> + |11>) + B * (|01> + |01>)

where A = Sqrt(1 – ErrorProbability^2) / sqrt(2) and B = Sqrt(ErrorProbability) / sqrt(2)
In the below I'll skip the precise expressions for A and B as their exact values are not important for the logic of the algorithm.

Now we somehow need to fix the value of the second state. As the initial pair is an entangled Bell pair we intend to use the first qubit along with an additional ancilla qubit. The code below fixes the qb2.

```
circuit.cx(qb1, ancilla[0])
circuit.cx(qb2, ancilla[0])
circuit.cx(ancilla[0], qb2)
```

To fix the sign flip we employ a similar idea combined with qubit rotation:

```
circuit.h(qb1)
circuit.h(qb2)

circuit.cx(qb1, ancilla[1])
circuit.cx(qb2, ancilla[1])
circuit.cx(ancilla[1], qb1)

circuit.h(qb1)
circuit.h(qb2)
```

Using the uncomputation test we can see the main qubits qb1[0] and qb2[0] are fixed.

Solution 2

This solution employs the error correction scheme in a more standard way. To avoid the dimension curse I had to use the built-in noise simulator, see the file **Solution2withBuiltInNoiseModel.py.**

The problem with employing the bit-flip correction in a straightforward manner is that the mixing induced but the Hadamard gate cannot be distinguished from the noise. To avoid that we would prefer to have the repeated qubits fully synchronized. I suggest then to implement entangling prior to the error step as below:

```
# We need the repeated qubits to be synchronize:
circ.h(qb1[0])
circ.cx(qb1[0], qb1[1])
circ.cx(qb1[0], qb1[2])
# Introducing error:
circ.id(qb1)
circ.id(qb2)

# # CNOT gate that entangles the two qubits
circ.cx(qb1, qb2)
```

Part 4. Discussion

Although the Solution I reproduces the Bell step I doubt that it is what was expected. To my mind it is not really an error-correction code as it will only work if the initial state is initialized as | 0 0 >.

The second solution is more cumbersome, more blunt and less precise but it works whatever the initial state