tomography

July 19, 2023

[1]:
```
from qiskit import QuantumCircuit, Aer, transpile
from qiskit.quantum_info import Statevector, state_fidelity, DensityMatrix
from qiskit.circuit.library import QuantumVolume
import numpy as np
import QuantumTomography as qKLib
import itertools
from qiskit.providers.fake_provider import FakeLimaV2
import matplotlib.pyplot as plt
```

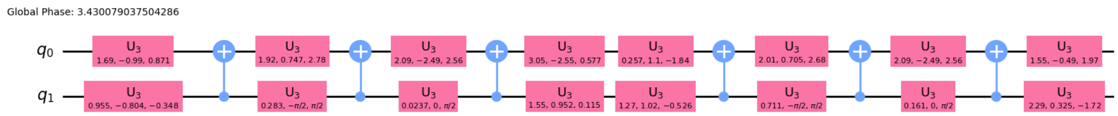
[2]:
```
class DensityMatrixRe():
    def __init__(self, circuit, backend=Aer.get_backend('aer_simulator'),␣
 ↪shots=100000):
        circuit.remove_final_measurements()
        self.circuit = circuit
        self.qnums = circuit.num_qubits
        self.backend = backend
        self.shots = shots

    def siMeasPiece(self, pauli, qpos):
        circ = QuantumCircuit(self.qnums)
        match pauli:
            case 'X':
                circ.h(qpos)
            case 'Y':
                circ.sdg(qpos)
                circ.h(qpos)
            case _:
                None
        return circ

    def siMeas(self, setting):
        circ = self.circuit.copy()
        for x in zip(setting, itertools.count(0, 1)):
            circ = circ.compose(self.siMeasPiece(*x))
        circ.measure_all()
        return circ
```

```python
    def tomoCircuits(self):
        self.measCirc = [self.siMeas(i) for i in itertools.product(['X', 'Y', 'Z'], repeat = self.qnums)]
        return None

    def simResult(self):
        self.tomoCircuits()
        simJob = self.backend.run(transpile(self.measCirc), shots = self.shots)
        result = simJob.result()
        self.counts = result.get_counts()
        tomoCoin = []
        for index in range(len(self.counts)):
            temp = []
            for item in [''.join(i) for i in itertools.product(['0', '1'], repeat=self.qnums)]:
                try:
                    temp = temp + [self.counts[index][item]]
                except KeyError:
                    temp = temp + [0]
            tomoCoin.insert(index, temp)
        return tomoCoin

    def dm(self):
        self.tomoCoin = np.array(self.simResult())
        self.tomo = qKLib.Tomography(nQ = self.qnums)
        self.tomo.conf['NQubits'] = self.qnums
        self.tomo.conf['NDetectors'] = 2
        self.tomo.conf['Crosstalk'] = np.eye(2**self.qnums, dtype = int)
        self.tomo.conf['Bellstate'] = 0
        self.tomo.conf['DoDriftCorrection'] = 0
        self.tomo.conf['DoAccidentalCorrection'] = 0
        self.tomo.conf['DoErrorEstimation'] = 0
        self.tomo.conf['Window'] = 0
        self.tomo.conf['Efficiency'] = 0
        dic = {'X': [1/np.sqrt(2), 1/np.sqrt(2)], 'Y': [1/np.sqrt(2), 1/np.sqrt(2)*1j], 'Z': [0, 1]}
        self.measurements = np.array([np.array([dic[i] for i in item]).flatten() \
                                      for item in itertools.product(['X', 'Y', 'Z'], repeat = self.qnums)])
        [rho, intensity, fval] = self.tomo.StateTomography(self.measurements, self.tomoCoin, method = 'LINEAR')
        self.tomo.conf['RhoStart'] = rho
        [rho, intensity, fval] = self.tomo.StateTomography(self.measurements, self.tomoCoin, method = 'MLE')
        return rho
```

[4]:

```python
testcirc = QuantumVolume(2, 2).decompose(reps=2)
testcirc.draw('mpl')
```

[4]:



[6]:

```python
#testcirc = QuantumCircuit(2)
#testcirc.h(0)
#testcirc.cnot(0, 1)
testdm = DensityMatrixRe(testcirc)
rho=testdm.dm() # reconstructed density matrix using qKLib
np.linalg.eig(rho)
```

[6]:

```
(array([6.00383817e-01+1.50225215e-18j, 3.63480436e-01+2.36190861e-17j,
        3.61357450e-02-8.35084443e-18j, 2.04389792e-09+1.83976831e-17j]),
 array([[ 0.52335856-0.33663679j,  0.32832753-0.34955648j,
         -0.35262399+0.33409831j,  0.09929248-0.37007895j],
        [-0.27397011-0.17600865j, -0.30104383+0.09291494j,
         -0.12250617+0.26528056j,  0.84221091+0.j        ],
        [ 0.67581084+0.j        , -0.56887518+0.03329994j,
          0.40851238-0.16813915j,  0.12520722-0.08787408j],
        [ 0.04899766-0.21820117j,  0.58823708+0.j        ,
          0.69534161+0.j        ,  0.28174342+0.20269509j]]))
```

[7]:

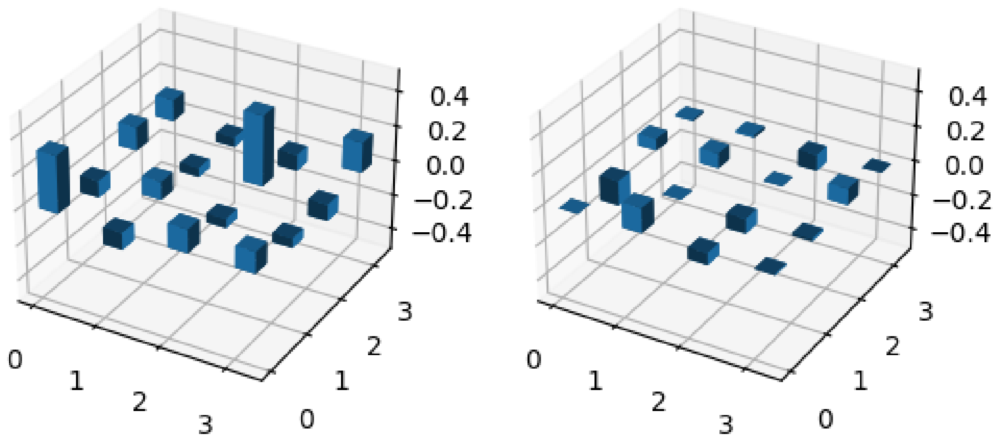
```python
print(state_fidelity(rho, Statevector(testdm.circuit).data))
print(Statevector(testdm.circuit).data) # ideal state vector
```

```
0.39635631042748964
[-0.45712215-0.26638187j  0.2773547 +0.65477807j  0.19802554-0.17528757j
 -0.31980419-0.20543983j]
```

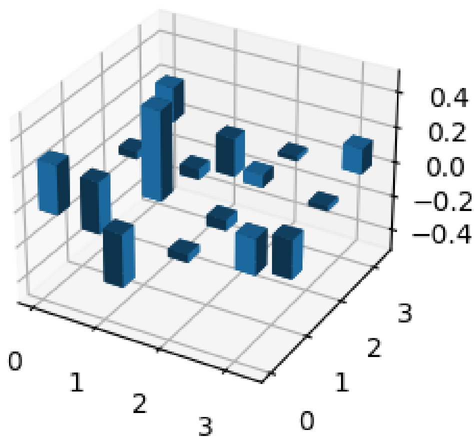
[8]:

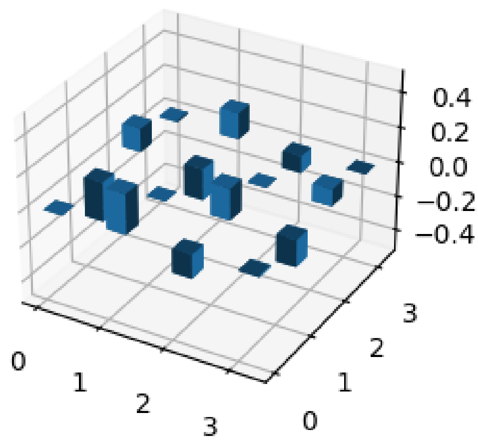
```python
# plot reconstructed density matrix
xx, yy = np.meshgrid(np.arange(2**2), np.arange(2**2))
X = xx.ravel()
Y = yy.ravel()
Z1 = rho.real.ravel()
Z2 = rho.imag.ravel()
height = np.zeros_like(Z1)
width = depth = 0.3
fig= plt.figure()
ax1 = fig.add_subplot(121, projection='3d')
ax2 = fig.add_subplot(122, projection='3d')
ax1.bar3d(X, Y, height, width, depth, Z1)
ax1.set_zlim(-0.5, 0.5)
```

```
ax2.bar3d(X, Y, height, width, depth, Z2)
ax2.set_zlim(-0.5, 0.5)
plt.show()
```



[9]:
```
# plot ideal density matrix
xx, yy = np.meshgrid(np.arange(2**2), np.arange(2**2))
X = xx.ravel()
Y = yy.ravel()
Z1 = DensityMatrix(testdm.circuit).data.real.ravel()
Z2 = DensityMatrix(testdm.circuit).data.imag.ravel()
height = np.zeros_like(Z1)
width = depth = 0.3
fig= plt.figure()
ax1 = fig.add_subplot(121, projection='3d')
ax2 = fig.add_subplot(122, projection='3d')
ax1.bar3d(X, Y, height, width, depth, Z1)
ax1.set_zlim(-0.5, 0.5)
ax2.bar3d(X, Y, height, width, depth, Z2)
ax2.set_zlim(-0.5, 0.5)
plt.show()
```





[10]:
```
testdm.tomoCoin # simulation of measurement results from XX to ZZ
```

[10]:
```
array([[ 2322, 20927, 14784, 61967],
       [14278, 64368,  2808, 18546],
       [ 9213, 69207,  7969, 13611],
       [ 4984, 18422, 12930, 63664],
       [10086, 68414,  7752, 13748],
       [16853, 61985,  1025, 20137],
       [13065, 10100, 22006, 54829],
       [30978, 47892,  4114, 17016],
       [27984, 50631,  6899, 14486]])
```

[ ]:
```
```