Li2+ 6 qubit_circuit simulation

January 20, 2020

[ ]:
```
from scipy.optimize import minimize
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister,IBMQ
import numpy as np
from qiskit import execute
from qiskit import BasicAer
backend = BasicAer.get_backend('qasm_simulator')
T=8192
IBMQ.load_accounts()
import random
```

[ ]:
```
def H1(theta):
    #create a Quantum Register called "q" with 10 qubits
    q = QuantumRegister(4)
    #create a Classical Register called "c" with 10 bits
    c = ClassicalRegister(4)
    qc = QuantumCircuit(q,c)

    qc.u1(theta[0], q[0])
    qc.u3(theta[1],-np.pi/2,np.pi/2, q[0])
    qc.u1(theta[2], q[0])
    qc.cx(q[0],q[1])
    qc.u1(theta[3], q[1])
    qc.u3(theta[4],-np.pi/2,np.pi/2, q[1])
    qc.u1(theta[5], q[1])
    qc.cx(q[1],q[0])
    qc.u1(theta[6], q[0])
    qc.u3(theta[7],-np.pi/2,np.pi/2, q[0])
    qc.u1(theta[8], q[0])
    qc.cx(q[0],q[2])
    qc.u1(theta[9], q[2])
    qc.u3(theta[10],-np.pi/2,np.pi/2, q[2])
    qc.u1(theta[11], q[2])
    qc.cx(q[2],q[0])
    qc.u1(theta[12], q[0])
    qc.u3(theta[13],-np.pi/2,np.pi/2, q[0])
    qc.u1(theta[14], q[0])
    qc.cx(q[0],q[3])
```

```
qc.u1(theta[15], q[3])
qc.u3(theta[16],-np.pi/2,np.pi/2, q[3])
qc.u1(theta[17], q[3])
qc.cx(q[3],q[0])
qc.u1(theta[18], q[1])
qc.u3(theta[19],-np.pi/2,np.pi/2, q[1])
qc.u1(theta[20], q[1])

qc.cx(q[1],q[2])
qc.u1(theta[21], q[2])
qc.u3(theta[22],-np.pi/2,np.pi/2, q[2])
qc.u1(theta[23], q[2])
qc.cx(q[2],q[1])
qc.u1(theta[24], q[1])
qc.u3(theta[25],-np.pi/2,np.pi/2, q[1])
qc.u1(theta[26], q[1])
qc.cx(q[1],q[3])
qc.u1(theta[27], q[3])
qc.u3(theta[28],-np.pi/2,np.pi/2, q[3])
qc.u1(theta[29], q[3])
qc.cx(q[3],q[1])
qc.u1(theta[30], q[2])
qc.u3(theta[31],-np.pi/2,np.pi/2, q[2])
qc.u1(theta[32], q[2])

qc.cx(q[2],q[3])
qc.u1(theta[33], q[3])
qc.u3(theta[34],-np.pi/2,np.pi/2, q[3])
qc.u1(theta[35], q[3])
qc.cx(q[3],q[2])
qc.u1(theta[36], q[2])
qc.u3(theta[37],-np.pi/2,np.pi/2, q[2])
qc.u1(theta[38], q[2])



#qc.z(q[0])
#qc.z(q[1])
#qc.z(q[2])
qc.z(q[3])


qc.measure(q[0], c[0])
qc.measure(q[1], c[1])
qc.measure(q[2], c[2])
qc.measure(q[3], c[3])
#print(qc)
```

```
    #print(i)
    shots = T       # Number of shots to run the program (experiment); maximum␣
↪is 8192 shots.
    max_credits = 10        # Maximum number of credits to spend on executions.

    job = execute(qc, backend=backend, shots=shots, max_credits=max_credits)
    results = job.result()
    counts = results.get_counts(qc)
    #print(counts11)
    tot_cnt=0
    if '0000' in list(counts):
        tot_cnt=tot_cnt+counts['0000']/T
        #print(tot_cnt)
    if '0001' in list(counts):
        tot_cnt=tot_cnt-counts['0001']/T
        #print(tot_cnt)
    if '0010' in list(counts):
        tot_cnt=tot_cnt+counts['0010']/T
        #print(tot_cnt)
    if '0011' in list(counts):
        tot_cnt=tot_cnt-counts['0011']/T
        #print(tot_cnt)
    if '0100' in list(counts):
        tot_cnt=tot_cnt+counts['0100']/T
        #print(tot_cnt)
    if '0101' in list(counts):
        tot_cnt=tot_cnt-counts['0101']/T
        #print(tot_cnt)
    if '0110' in list(counts):
        tot_cnt=tot_cnt+counts['0110']/T
        #print(tot_cnt)
    if '0111' in list(counts):
        tot_cnt=tot_cnt-counts['0111']/T
        #print(tot_cnt)
    if '1000' in list(counts):
        tot_cnt=tot_cnt+counts['1000']/T
        #print(tot_cnt)
    if '1001' in list(counts):
        tot_cnt=tot_cnt-counts['1001']/T
        #print(tot_cnt)
    if '1010' in list(counts):
        tot_cnt=tot_cnt+counts['1010']/T
        #print(tot_cnt)
    if '1011' in list(counts):
        tot_cnt=tot_cnt-counts['1011']/T
        #print(tot_cnt)
    if '1100' in list(counts):
```

```
        tot_cnt=tot_cnt+counts['1100']/T
        #print(tot_cnt)
    if '1101' in list(counts):
        tot_cnt=tot_cnt-counts['1101']/T
        #print(tot_cnt)
    if '1110' in list(counts):
        tot_cnt=tot_cnt+counts['1110']/T
        #print(tot_cnt)
    if '1111' in list(counts):
        tot_cnt=tot_cnt-counts['1111']/T
        #print(tot_cnt)

    return tot_cnt

angles = [0 , -np.pi , -np.pi/2 , np.pi , np.pi/2]
def rand(num):
    res = []

    for j in range(num):
        res.append(random.sample(angles, k = 1))

    return res
theta0= rand(39)

result = minimize(H1, theta0, method='powell',options={'xtol': 1e-8 , 'disp':
 ↪True})
print(result, H1)
```

[ ]:
```
theta = [ 1.48690748, -2.27519207, -1.83019546, -3.56768217, -0.68024792,
        1.09447198, -4.08998329,  6.53054946, -0.55003959,  3.3244461 ,
        5.80669457,  4.83161712, -4.36467964, -1.82765036,  6.8313614 ,
        4.61552484,  3.13003084,  7.99003881,  0.9168758 ,  0.45324498,
        0.20296852,  5.28187802,  1.48205029,  1.7238057 , -2.05887548,
        5.81436504,  7.40819899,  1.91595123,  1.16516512,  3.85714742,
        5.18191127,  6.40844816,  3.02776418,  5.16297741, -1.27730928,
        4.61802716,  5.76018672,  2.78218976,  2.66262189]
#create a Quantum Register called "q" with 10 qubits
q = QuantumRegister(4)
#create a Classical Register called "c" with 10 bits
c = ClassicalRegister(4)
qc = QuantumCircuit(q,c)

qc.u1(theta[0], q[0])
qc.u3(theta[1],-np.pi/2,np.pi/2, q[0])
qc.u1(theta[2], q[0])
qc.cx(q[0],q[1])
qc.u1(theta[3], q[1])
```

```
qc.u3(theta[4],-np.pi/2,np.pi/2, q[1])
qc.u1(theta[5], q[1])
qc.cx(q[1],q[0])
qc.u1(theta[6], q[0])
qc.u3(theta[7],-np.pi/2,np.pi/2, q[0])
qc.u1(theta[8], q[0])
qc.cx(q[0],q[2])
qc.u1(theta[9], q[2])
qc.u3(theta[10],-np.pi/2,np.pi/2, q[2])
qc.u1(theta[11], q[2])
qc.cx(q[2],q[0])
qc.u1(theta[12], q[0])
qc.u3(theta[13],-np.pi/2,np.pi/2, q[0])
qc.u1(theta[14], q[0])
qc.cx(q[0],q[3])
qc.u1(theta[15], q[3])
qc.u3(theta[16],-np.pi/2,np.pi/2, q[3])
qc.u1(theta[17], q[3])
qc.cx(q[3],q[0])
qc.u1(theta[18], q[1])
qc.u3(theta[19],-np.pi/2,np.pi/2, q[1])
qc.u1(theta[20], q[1])

qc.cx(q[1],q[2])
qc.u1(theta[21], q[2])
qc.u3(theta[22],-np.pi/2,np.pi/2, q[2])
qc.u1(theta[23], q[2])
qc.cx(q[2],q[1])
qc.u1(theta[24], q[1])
qc.u3(theta[25],-np.pi/2,np.pi/2, q[1])
qc.u1(theta[26], q[1])
qc.cx(q[1],q[3])
qc.u1(theta[27], q[3])
qc.u3(theta[28],-np.pi/2,np.pi/2, q[3])
qc.u1(theta[29], q[3])
qc.cx(q[3],q[1])
qc.u1(theta[30], q[2])
qc.u3(theta[31],-np.pi/2,np.pi/2, q[2])
qc.u1(theta[32], q[2])

qc.cx(q[2],q[3])
qc.u1(theta[33], q[3])
qc.u3(theta[34],-np.pi/2,np.pi/2, q[3])
qc.u1(theta[35], q[3])
qc.cx(q[3],q[2])
qc.u1(theta[36], q[2])
qc.u3(theta[37],-np.pi/2,np.pi/2, q[2])
```

```
qc.u1(theta[38], q[2])



#qc.z(q[0])
#qc.z(q[1])
#qc.z(q[2])
qc.z(q[3])


qc.measure(q[0], c[0])
qc.measure(q[1], c[1])
qc.measure(q[2], c[2])
qc.measure(q[3], c[3])
#print(qc)
#print(i)
shots = T       # Number of shots to run the program (experiment); maximum is␣
 ↪8192 shots.
max_credits = 10         # Maximum number of credits to spend on executions.

job_hpc = execute(qc, backend=Aer.get_backend('qasm_simulator'), shots=shots,␣
 ↪max_credits=max_credits)
result_hpc = job_hpc.result()
counts = result_hpc.get_counts(qc)
samples = result_hpc.get_memory()
print(samples)

print(counts)
```

[ ]:
```
#IIZI
from qiskit.visualization import plot_histogram
plot_histogram(counts)
```

[ ]:
```
```