Mentorship-Task4

February 23, 2020

[1]:
```
from pyquil import Program, get_qc
import pyquil.api as api
from pyquil.gates import *
from pyquil.paulis import *
from scipy.optimize import minimize
from functools import partial
```

0.0.1 Useful Matrices

$$\frac{I+Z}{2} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad \frac{I-Z}{2} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \qquad \frac{X+\iota Y}{2} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \qquad \frac{I-\iota Y}{2} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

0.0.2 Given Hamiltonian

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

0.0.3 Decomposing H:

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$H = \left(\frac{(I+Z)}{2} \otimes \frac{(Z-I)}{2}\right) + \left(\frac{(Z-I)}{2} \otimes \frac{(I+Z)}{2}\right) + \left(\frac{(X+\iota Y)}{2} \otimes \frac{(X-\iota Y)}{2}\right) + \left(\frac{(X-\iota Y)}{2} \otimes \frac{(X+\iota Y)}{2}\right)$$

$$H = \frac{XX + YY + ZZ - II}{2}$$

[2]:
```
def ansatz(params): # Returns a parameterized circuit
    return Program(RX(params[0], 0), RX(params[1], 1), CNOT(0, 1),␣
↪RZ(params[2], 0), RZ(params[3], 1))
```

[3]:
```
a = 0.5 * sX(0) * sX(1)
b = 0.5 * sY(0) * sY(1)
c = 0.5 * sZ(0) * sZ(1)
d = -0.5 * ID() * ID()

# Construct a sum of Pauli terms.
Hamiltonian = a + b + c + d
```

[4]:
```
def expec_meas(prog, index_list, qc, shots):
    # Doing full measurements on given qubits
    program = Program()
    program += prog
    ro = program.declare('ro', 'BIT', max(index_list) + 1)
    program += [MEASURE(qubit, r) for qubit, r in␣
↪zip(list(range(max(index_list) + 1)), ro)]
    program.wrap_in_numshots_loop(shots)
    executable = qc.compile(program)
    results = qc.run(executable)

    # Create a frequency dictionary
    freq = {}
    result = list(map(tuple, results))
    for items in result:
        freq[items] = result.count(items)

    # Find parity for each possible outcome
    # Eg => Z = P(0) - P(1)
    # Eg => ZZ = P(00) + P(11) - P(01) - P(10)
    mask = 0
    for qb in index_list:
        mask |= 1 << qb
    parity = {state:1 if bin(mask & state).count("1") % 2 == 0 else -1 for␣
↪state in
              [int("".join([str(x) for x in y[::-1]]), 2) for y in freq.keys()]}

    exp_res = 0
    for bit, count in freq.items():
        bit_int = int("".join([str(x) for x in bit[::-1]]), 2)
        exp_res += float(count) * parity[bit_int]

    exp_res /= shots # To calculate mean i.e. probability
    return exp_res
```

[7]:
```
def cost(angles, hamil, qc, shots):
    prog = ansatz(angles) # Building the Ansatz
    ham_expec = 0
```

```
    term_expec = np.zeros(len(Hamiltonian))  # Stores expectation of each pauli␣
↪terms
    for ind, term in enumerate(hamil):
        meas_basis = Program()
        qubits_measure_idx = []
        if is_identity(term):
            term_expec[ind] = term.coefficient.real #Expectation for the ID term
        else:
            for idx, gate in term.operations_as_set(): # Makes circuit for␣
↪measurement
                qubits_measure_idx.append(idx)
                if gate == 'X': #X basis measurement
                    meas_basis.inst(RY(-0.5 * np.pi, idx))
                elif gate == 'Y': #Y basis measurement
                    meas_basis.inst(RX(0.5 * np.pi, idx))
            qubits_measure_idx.sort()
            expec_res = expec_meas(prog+meas_basis, qubits_measure_idx, qc,␣
↪shots)
            term_expec[ind] = term.coefficient.real * expec_res
    print(np.sum(term_expec))
    return np.sum(term_expec)
```

[13]:
```
qc = get_qc("2q-qvm")
x0 = np.array([0,0,0,0])
vqe = partial(cost, hamil=Hamiltonian, qc=qc, shots=1024)
fun = minimize(vqe, x0, method='COBYLA') # This is the hybrid part
print(fun)
```

```
0.0107421875
-0.00390625
-0.275390625
-0.0634765625
-0.3916015625
-1.12890625
-1.259765625
-1.595703125
-0.5595703125
-1.19140625
-1.8466796875
-1.59375
-1.6689453125
-1.787109375
-1.6494140625
-1.8232421875
-1.9423828125
-1.9873046875
-1.9228515625
```

```
-1.96875
-1.923828125
-1.9951171875
-2.0
-1.9970703125
-1.990234375
-1.9990234375
-1.9921875
-1.9990234375
-1.9970703125
-1.9990234375
-1.9990234375
-1.998046875
-2.0
-2.0
-1.9970703125
-2.0
-1.998046875
-1.9990234375
-2.0
-2.0
-1.998046875
-1.9990234375
-1.9970703125
-2.0
-1.998046875
-1.9990234375
-1.9990234375
-2.0
-1.9990234375
-1.9990234375
-2.0
-2.0
     fun: -2.0
   maxcv: 0.0
 message: 'Optimization terminated successfully.'
    nfev: 52
  status: 1
 success: True
       x: array([ 1.60298664,  3.13586389, -0.48670763,  1.10937732])
```

[22]:
```
fun['x'] # Optimized angles
```

[22]:
```
array([ 1.60298664,  3.13586389, -0.48670763,  1.10937732])
```

[34]:
```
fun['fun'] # Final value i.e. our lowest eigenvalue
```

[34]: -2.0

[14]:
```
from pyquil.api import WavefunctionSimulator # For cross checking purposes
wf_sim = WavefunctionSimulator()
p = ansatz(fun['x'])
wf = wf_sim.wavefunction(p)
print(wf)
```

```
(0.0018967678-0.0006103798j)|00> + (-0.5015168858+0.5143626704j)|01> +
(0.4980660633-0.4856272732j)|10> + (0.0006303513-0.0019588296j)|11>
```

[27]:
```
wf.amplitudes
```

[27]:
```
array([ 0.00189677-0.00061038j, -0.50151689+0.51436267j,
        0.49806606-0.48562727j,  0.00063035-0.00195883j])
```

[17]:
```
Ham_mat = np.array([[0,0,0,0],[0,-1,1,0],[0,1,-1,0],[0,0,0,0]])
```

[20]:
```
np.matmul(Ham_mat,wf.amplitudes)
```

[20]:
```
array([ 0.        +0.j        ,  0.99958295-0.99998994j,
       -0.99958295+0.99998994j,  0.        +0.j        ])
```

0.0.4 Final Check

Calculating the following:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0.00189677 - 0.00061038\imath \\ -0.50151689 + 0.51436267\imath \\ , 0.49806606 - 0.48562727\imath \\ , 0.00063035 - 0.00195883\imath \end{bmatrix} + 2 \times \begin{bmatrix} 0.00189677 - 0.00061038\imath \\ -0.50151689 + 0.51436267\imath \\ , 0.49806606 - 0.48562727\imath \\ , 0.00063035 - 0.00195883\imath \end{bmatrix}$$

[32]:
```
np.abs(np.matmul(Ham_mat,wf.amplitudes) - fun['fun']*wf.amplitudes) #␣
↪Characterstic equations
```

[32]:
```
array([0.00398512, 0.02894186, 0.02894186, 0.00411551])
```