BirminghamInterviewExercise

February 2, 2016

1 Birmingham Interview Exercise

The spin-1 matrices have the following 3x3 representations

```
In [167]: Sz
```

Out[167]:

Quantum object: dims = [[3], [3]], shape = [3, 3], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -1.0 \end{pmatrix}$$

```
In [168]: Sy
```

Out[168]:

Quantum object: dims = [[3], [3]], shape = [3, 3], type = oper, isherm = True

$$\begin{pmatrix} 0.0 & -0.707j & 0.0 \\ 0.707j & 0.0 & -0.707j \\ 0.0 & 0.707j & 0.0 \end{pmatrix}$$

```
In [169]: Sx
```

Out[169]:

Quantum object: dims = [[3], [3]], shape = [3, 3], type = oper, isherm = True

$$\begin{pmatrix} 0.0 & 0.707 & 0.0 \\ 0.707 & 0.0 & 0.707 \\ 0.0 & 0.707 & 0.0 \end{pmatrix}$$

The Hamiltonian

$$H = J_x S_1^x S_2^x + J_y S_1^y S_2^y + J_z S_1^z S_2^z$$

is the weighted sum of tensor products of spin operators, which in the spin basis can be represented as a 9x9 matrix

```
In [308]: def H(Jx, Jy, Jz):
              return Jx*qt.tensor(Sx, Sx) + Jy*qt.tensor(Sy, Sy) + Jz*qt.tensor(Sz, Sz)
```

2 Jx = Jy = Jz = 1

```
In [309]: H1 = H(1, 1, 1)
          H1
```

Out[309]:

Quantum object: dims = [[3, 3], [3, 3]], shape = [9, 9], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -1.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & -1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

The matrix has dimension 9x9 and is not rank-deficient, and thus has 9 eigenvalues and eigenvectors. The groundstate is the eigenvector corresponding to the lowest eigenvalue (this uses a procedure from QuTiP which uses the Cholesky solver from the Numpy Library and returns the lowest energy eigenvalue and vector)

```
In [342]: ground_state_energy_1, ground_state_ket_1 = H1.groundstate()
          np.isclose(ground_state_energy_1, -2)
```

```
Out[342]: True
```

By inspection, the ground state eigenvector can be expressed in terms of the tensor products of N=3 spin states, where the expansion (schmidt) coefficients are

$$p_1 = -\sqrt{\frac{1}{3}}, p_2 = \sqrt{\frac{1}{3}}, p_3 = -\sqrt{\frac{1}{3}}$$

and the vectors are

$$v_1 = |0, 2\rangle, v_2 = |1, 1\rangle, v_3 = |2, 0\rangle$$

```
In [316]: ground_state_ket_1
```

Out[316]:

Quantum object: dims = [[3, 3], [1, 1]], shape = [9, 1], type = ket

$$\begin{pmatrix} 0.0 \\ 0.0 \\ -0.577 \\ 0.0 \\ 0.577 \\ 0.0 \\ -0.577 \\ 0.0 \\ 0.0 \end{pmatrix}$$

3 Jx = 0.1, Jy = 0.2, Jz = 1

The non-maximally entangled case

```
In [410]: H2 = H(0.1, 0.2, 1.0)
          H2
```

```
Out[410]:
```

Quantum object: dims = [[3, 3], [3, 3]], shape = [9, 9], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & -0.050 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.150 & 0.0 & -0.050 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -1.0 & 0.0 & 0.150 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.150 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.050 & 0.0 \\ -0.050 & 0.0 & 0.150 & 0.0 & 0.0 & 0.0 & 0.150 & 0.0 & -0.050 \\ 0.0 & -0.050 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.150 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.150 & 0.0 & -1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -0.050 & 0.0 & 0.150 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & -0.050 & 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

```
In [411]: g2 = H2.groundstate()
          ground_state_energy_2 = g2[0]
          ground_state_energy_2
```

```
Out[411]: -1.0432364160774361
```

```
In [412]: ground_state_ket_2 = g2[1]
          ground_state_ket_2.full()
```

```
Out[412]: array([[ 0.00488704+0.j],
                 [ 0.00000000+0.j],
                 [-0.69284525+0.j],
                 [ 0.00000000+0.j],
                 [ 0.19970764+0.j],
                 [ 0.00000000+0.j],
                 [-0.69284525+0.j],
                 [ 0.00000000+0.j],
                 [ 0.00488704+0.j]])
```

4 Schmidt Decomposition

The schmidt decomposition is related to the SVD decomposition (for a finite dimensional Hilbert space) as follows

$$|\psi\rangle = \sum_i \sum_j C_{ij} |a\rangle_i \otimes |b\rangle_j = \sum_i \sqrt{p_i} |\alpha\rangle \otimes |\beta\rangle_i$$

where the SVD of the matrix C_{ij} gives vectors U, Σ, V such that

$$C = U \Sigma V^\dagger$$

and

$$|\alpha\rangle_i = U|a\rangle_i$$

$$|\beta\rangle_i = V^*|b\rangle_i$$

$$\Sigma_{ii} = \sqrt{p_i}$$

```
In [417]: ## Find the canonical expansion coefficients by inspection
          # System 1
          -1/(np.sqrt(3))*(
              qt.tensor(qt.basis(3, 0), qt.basis(3, 2)) -
              qt.tensor(qt.basis(3, 1), qt.basis(3, 1)) +
              qt.tensor(qt.basis(3, 2), qt.basis(3, 0))) ;
```

```
# System 2
0.00488704 * qt.tensor(
    qt.basis(3, 0), qt.basis(3, 0)) - 0.69284525 * qt.tensor(
    qt.basis(3, 2), qt.basis(3, 0)) + 0.19970764 * qt.tensor(
    qt.basis(3, 1), qt.basis(3, 1)) - 0.69284525 * qt.tensor(
    qt.basis(3, 0), qt.basis(3, 2)) + 0.00488704 * qt.tensor(
    qt.basis(3, 2), qt.basis(3, 2)) ;

# canonical basis vectors for n dim system
dim = 3
basis = [qt.basis(dim, i).full() for i in range(dim)]
```

In [414]:
```
# from inspection of system 1
C_1 = np.array(
    [[0, 0, -np.sqrt(1/3)],
     [0, np.sqrt(1/3), 0],
     [-np.sqrt(1/3), 0, 0]])
res(C_1).T

# perform svd
U_1, sig_1, Vst_1 = LA.svd(C_1)
beta_1 = [Vst_1.T.dot(vec) for vec in basis]
alpha_1 = [U_1.dot(vec) for vec in basis]
schmidts_1 = [l for l in sig_1]

## expansion in terms of the schmidt coefficients
## reproduces the original state vector (to machine precision)
assert not (sum(
        [schmidts_1[i]*
         qt.tensor(qt.Qobj(alpha_1[i]), qt.Qobj(beta_1[i])) for i in range(dim)]) -
            ground_state_ket_1).full().any()
## new vector is normalised
assert np.sum([schmidt**2 for schmidt in schmidts_1]) == 1

schmidts_1
```

Out[414]:
```
[0.57735026918962573, 0.57735026918962573, 0.57735026918962573]
```

In [418]:
```
# from inspection of system 2
C_2 = np.array([[0.00488704, 0, -0.69284525],
                [0, 0.19970764, 0],
                [-0.69284525, 0, 0.00488704]])

# perform decomposition, produce new basis vectors and schmidt coefficients
prec = 8
U_2, sig2, Vst_2 = LA.svd(C_2)
beta_2 = [np.around(Vst_2.T.dot(vec), decs) for vec in basis]
alpha_2 = [np.around(U_2.dot(vec), decs) for vec in basis]
schmidts_2 = [np.around(l, decs) for l in sig_2]

## The expansion in terms of the schmidt coefficients
## reproduces the original state vector to within 10**8 tolerance
assert not (sum(
        [schmidts_2[i]*qt.tensor(
                qt.Qobj(alpha_2[i]), qt.Qobj(beta_2[i])) for i in range(dim)]) -
```

```
                ground_state_ket_2).tidyup(10**-prec).full().any()
        ## new vector is normalised
        assert np.isclose(np.sum([schmidt**2 for schmidt in schmidts_2]), 1)

        schmidts_2
```

Out[418]: [0.69773229000000003, 0.68795821000000001, 0.19970763999999999]

5 Von Neumann entropies

The von neumann entropy in terms of the Schmidt coefficients

$$S(p) = -\Sigma \rho_i ln \rho_i$$

In [419]:
```
def von_neumann_entropy(nums):
    '''takes schmidt coefficients and returns von-neumann entropy'''
    return -np.sum([num**2*np.log(num**2) for num in nums])
```

In [420]:
```
system_1_von_neumann_entropy = von_neumann_entropy(schmidts_1)
system_2_von_neumann_entropy = von_neumann_entropy(schmidts_2)
```

In [421]:
```
system_1_von_neumann_entropy, np.log(3)
```

Out[421]: (1.0986122886681096, 1.0986122886681098)

In [422]:
```
system_2_von_neumann_entropy
```

Out[422]: 0.83297935252154198

System 1 is maximally entangled, and has the corresponding entanglement entropy log(N), w/ N=3 the number of schmidt coefficients

In [423]:
```
# Imports, setup, tools
import numpy as np
import qutip as qt
import numpy.linalg as LA
import cmath

Sx = qt.Qobj((1/np.sqrt(2))*np.array([[0, 1, 0], [1, 0, 1], [0, 1, 0]]))
Sy = qt.Qobj((1/(np.sqrt(2)*1j))*np.array([[0, 1, 0], [-1, 0, 1], [0, -1, 0]]))
Sz = qt.Qobj(np.array([[1, 0, 0], [0, 0, 0], [0, 0, -1]]))

def re_shuffle(array, m, n):
    return np.asarray([[np.trace(np.outer(basis[i], basis[j]).T.dot(array)) for i in range(m)]

def res_shuffle(array, m, n):
    return np.asarray([[res(np.outer(basis[i], basis[j])).dot(res(array)) for i in range(m)] 

def res(array):
    ret_list = []
    for row in array:
        ret_list+=list(row)
    return np.asarray([ret_list])
def vec(array):
    ret_list = []
    for col in array.T:
        ret_list+=list(col.T)
    return np.asarray([ret_list])
```