Quantum Many Body Physics – Final Project

1. Number of basis states:

$$N=3, M=2 \rightarrow |30\rangle, |03\rangle, |21\rangle, |12\rangle$$
$$N=2, M=3 \rightarrow |002\rangle, |020\rangle, |200\rangle, |110\rangle, |101\rangle, |011\rangle$$

In general, there are N+M-1 positions. We want to fill N of these positions, hence:

$$\binom{N+M-1}{N} = C(N+M-1, N) = \frac{(N+M-1)!}{N!(M-1)!}$$

2. Renyi Entropy:

$$|\psi(0)\rangle = |11\rangle = (0,1,0), |\psi(t)\rangle = \exp\left[\frac{-it\hat{H}}{\hbar}\right]|\psi(0)\rangle$$

$\hat{H}$ in matrix form from basis states:

$$\hat{H}|02\rangle = -J\sqrt{2}|11\rangle + U|02\rangle$$
$$\hat{H}|11\rangle = -J\sqrt{2}|02\rangle - J\sqrt{2}|20\rangle$$
$$\hat{H}|20\rangle = -J\sqrt{2}|11\rangle + U|20\rangle$$
$$\hat{H} = \begin{bmatrix} U & -J\sqrt{2} & 0 \\ -J\sqrt{2} & 0 & -J\sqrt{2} \\ 0 & -J\sqrt{2} & U \end{bmatrix}$$

Setting U = J = 1:

$$|\hat{H} - \mathbb{I}E| = \begin{vmatrix} 1-E & -\sqrt{2} & 0 \\ -\sqrt{2} & -E & -\sqrt{2} \\ 0 & -\sqrt{2} & 1-E \end{vmatrix} = 0$$
$$\rightarrow (1-E)(-E+E^2-2)+2E-2 \;=\; (1-E)(-E+E^2-4) \;=\; 0$$
$$\rightarrow \;\; E_1=1, \;\; E_2=\frac{1}{2}+\frac{\sqrt{17}}{2}, \;\; E_3=\frac{1}{2}-\frac{\sqrt{17}}{2}$$

This leads to eigenstates:

$$\bar{\psi}_1 = \frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}, \bar{\psi}_2 = \frac{2}{\sqrt{17-\sqrt{17}}}\begin{pmatrix} 1 \\ -\frac{1}{2}\sqrt{9-\sqrt{17}} \\ 1 \end{pmatrix}, \bar{\psi}_3 = \frac{2}{\sqrt{17+\sqrt{17}}}\begin{pmatrix} 1 \\ \frac{1}{2}\sqrt{9+\sqrt{17}} \\ 1 \end{pmatrix}$$

And a time evolution:

$$|\psi(t)\rangle = \sum_i exp\left(\frac{-iE_i t}{\hbar}\right)|\bar{\psi_i}\rangle\langle\bar{\psi_i}|\psi(0)\rangle$$

$$= e^{\frac{-it}{\hbar}}|\bar{\psi_1}\rangle\langle\bar{\psi_1}|\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + e^{\frac{-it(\frac{1}{2}+\frac{\sqrt{17}}{2})}{\hbar}}|\bar{\psi_2}\rangle\langle\bar{\psi_2}|\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + e^{\frac{-it(\frac{1}{2}-\frac{\sqrt{17}}{2})}{\hbar}}|\bar{\psi_3}\rangle\langle\bar{\psi_3}|\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} e^{-i\alpha}\gamma a + e^{-i\beta}\delta b \\ e^{-i\alpha}\gamma a^2 + e^{-i\beta}\delta b^2 \\ e^{-i\alpha}\gamma a + e^{-i\beta}\delta b \end{pmatrix}$$

Where:

$$\alpha = \frac{t(1+\sqrt{17})}{2\hbar}, \beta = \frac{t(1-\sqrt{17})}{2\hbar}$$

$$a = -\frac{1}{2}\sqrt{9-\sqrt{17}}, b = \frac{1}{2}\sqrt{9+\sqrt{17}}$$

$$\gamma = \frac{4}{17-\sqrt{17}}, \delta = \frac{4}{17+\sqrt{17}}$$

This leads to the reduced density matrix:

$$\hat{\rho_A} = \begin{pmatrix} (e^{-i\alpha}x + e^{-i\beta}y)(e^{i\alpha}x + e^{i\beta}y) & 0 & 0 \\ 0 & (e^{-i\alpha}f + e^{-i\beta}g)(e^{i\alpha}f + e^{i\beta}g) & 0 \\ 0 & 0 & (e^{-i\alpha}x + e^{-i\beta}y)(e^{i\alpha}x + e^{i\beta}y) \end{pmatrix}$$

$$= \begin{pmatrix} 2xycos(\alpha-\beta) + x^2 + y^2 & 0 & 0 \\ 0 & 2fgcos(\alpha-\beta) + f^2 + g^2 & 0 \\ 0 & 0 & 2xycos(\alpha-\beta) + x^2 + y^2 \end{pmatrix}$$

Where we have set:

$$x = \gamma a, y = \delta b, f = \gamma a^2 \;\; g = \delta b^2$$

And a squared reduced density matrix:

$$\hat{\rho_A^2} = \begin{pmatrix} A & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & A \end{pmatrix}$$

Where:

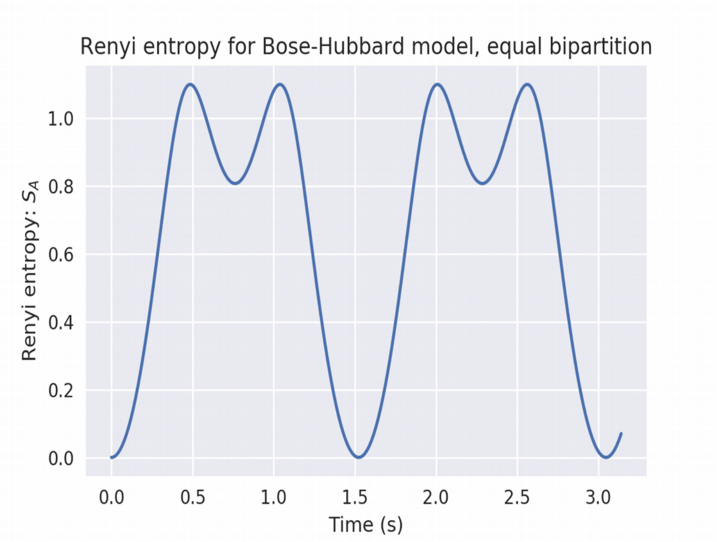
$$A = 4x^2y^2cos^2(\frac{t\sqrt{17}}{\hbar}) + 4(x^3y + y^3x)cos(\frac{t\sqrt{17}}{\hbar}) + 2x^2y^2 + x^4 + y^4$$

$$B = 4f^2g^2cos^2(\frac{t\sqrt{17}}{\hbar}) + 4(f^3g + g^3f)cos(\frac{t\sqrt{17}}{\hbar}) + 2f^2g^2 + f^4 + g^4$$

Calculating the Renyi entropy as $S(t) = -\ln(Tr(\rho_A^2(t)))$:

$$S(t) = -ln\left[(8x^2y^2 + 4f^2g^2)cos^2\left(\frac{t\sqrt{17}}{\hbar}\right) + 4(2x^3y + 2y^3x + f^3g + g^3f)cos\left(\frac{t\sqrt{17}}{\hbar}\right) + const.\right]$$

Which looks like:

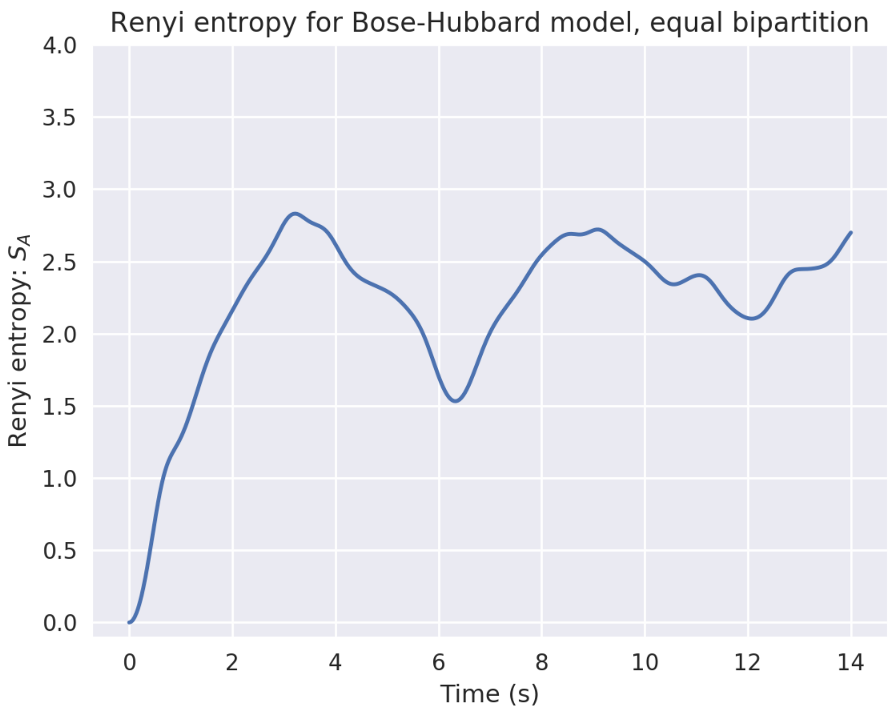


3. Simulation:
Simulation of the Bose-Hubbard model for a system of 6 bosons, equally bipartitioned.
The parameters are J = 0.64, U = 1, N=M=6
The initial state is: $|111111\rangle$

For this system, the 4 lowest energies are:

-4.71081856	-4.11963126	-3.6680174	-3.41149995

The resultant plot for Renyi entropy over time is:



See overleaf for code.

```python
# -*- coding: utf-8 -*-
"""
author: Sam Spillard
python script to complete final project of Quantum Many-Body Physics module
Simulate experiment by Kaufman et al. 2016

Python version: 3.5.4
"""

from scipy.special import factorial as fact
import scipy as sp
import numpy as np
import itertools
from copy import deepcopy
import matplotlib.pyplot as plt
#import seaborn as sns
#sns.set()

class StateObj:
    """
    Class to keep track of state vector, prefactor and type
    idx is a parameter that keeps basis states in the correct order according
    to the integer representation of the vector.
    Includes creation, annihilation, number, time evolution, rdm and entropy
    operators, deepcopy'd to prevent modifying basis states.
    """
    # Initialise attributes
    def __init__(self, init_vec, idx, _type):
        if(type(init_vec) != np.ndarray):
            raise TypeError('init_vec should be a numpy.ndarray')
        self.vector = init_vec
        self.idx = idx
        self.type = _type
        self.prefactor = 1
    # Creation operator
    def create(self, index, N):
        trans = deepcopy(self)
        if(np.sum(trans.vector) > N):
            trans.prefactor = 0
        else:
            trans.vector[index] += 1
            trans.prefactor *= np.sqrt(trans.vector[index])
        return trans
    # Annihilation operator
    def destroy(self, index):
        trans = deepcopy(self)
        if(trans.vector[index] == 0):
            trans.prefactor = 0
        else:
            trans.prefactor *= np.sqrt(trans.vector[index])
            trans.vector[index] -= 1
        return trans
    # Number operator
    def num(self, index):
        return self.vector[index]
    # Time Evolution
    def tevolve(self, hamMatrix, t):
        if(self.type == 'boson'):
            raise TypeError('State should not be in boson format')
        trans = deepcopy(self)
        expMat = sp.linalg.expm(-1j * hamMatrix * t)
```

```python
        trans.vector = np.dot(expMat, self.vector)
        return trans
    # Reduced Density Matrix
    def rdm(self, N, M, basis, ASIZE=1):
        # Every available a state
        a_basis = np.asarray([np.asarray(i) for i in itertools.product(
                range(N+1), repeat=ASIZE)])
        # Every available b state
        b_basis = np.asarray([np.asarray(i) for i in itertools.product(
                range(N+1), repeat=(M-ASIZE))])
        # Initialise c_matrix as zeros
        ALEN, BLEN = len(a_basis), len(b_basis)
        c_matrix = np.zeros((ALEN, BLEN), dtype=complex)
        for i in range(len(self.vector)):
            a_vec = basis[i].vector[:ASIZE]
            # Get matrix indices for entry
            for j in range(ALEN):
                if(np.all(a_basis[j] == a_vec)):
                    a_idx = j
                    break
            b_vec = basis[i].vector[ASIZE:]
            for j in range(BLEN):
                if(np.all(b_basis[j] == b_vec)):
                    b_idx = j
                    break

            c_matrix[a_idx, b_idx] = self.vector[i]*self.prefactor

        RDM = np.dot(c_matrix, c_matrix.conj().T)

        return RDM
    # Entropy
    def entropy(self, N, M, basis, ASIZE):
        rdm = self.rdm(N, M, basis, ASIZE)
        vals, vecs = sp.linalg.eigh(np.dot(rdm, rdm))
        return -np.log(np.sum(vals))


def getBasisStates(N, M):
    """
    Function to get a list of basis states, ordered by the integer
    representation of the state vector.
    """
    x = itertools.product(range(N + 1), repeat=M)
    basis = [np.asarray(i, dtype=int) for i in x if np.sum(i)==N]
    states = np.asarray([StateObj(basis[i], i, 'boson') for i in
                         range(len(basis))])

    return states

def actHam(state, N, J, U):
    """
    Act hamiltonian on a state, as a series of Creation, Annihilation and
    Number operators.
    Multiply by appropriate parameters, J, U.
    Returns an array of new states with appropriate prefactors.
    """
    t1, t2 = [], []
    # First term
    for i in range(len(state.vector)-1):
        t1.append(state.create(i+1, N).destroy(i))
        t1.append(state.create(i, N).destroy(i+1))
    # Second term
    for i in range(len(state.vector)):
```

```python
        PREFACTOR = state.num(i) * (state.num(i) - 1)
        temp3 = deepcopy(state)
        temp3.prefactor *= (PREFACTOR*U/2)
        t2.append(temp3)

    for state in t1:
        state.prefactor *= (-1 * J)

    return np.r_[t1, t2]

def getHamMatrix(N, M, J, U):
    """
    Get hamiltonian matrix by acting hamiltonian on each basis state.
    """
    basis = getBasisStates(N, M)
    ham_matrix = np.zeros((len(basis), len(basis)))
    for state in basis:
        # Act ham on each basis state
        acted = actHam(state, N, J, U)
        for x in acted:
            for b in basis:
                # Find the matrix location of the 'acted' state and enter into
                # Hamiltonian matrix.
                if(np.all(x.vector == b.vector)):
                    ham_matrix[state.idx][b.idx] += x.prefactor
    # Return ham matrix and basis
    return ham_matrix, basis

def getInitialState(N, M):
    """
    Get initial state vector from user.
    """
    LENGTH = int((fact(N+M-1))/(fact(N)*fact(M-1)))
    temp_basis = np.asarray([x.vector for x in getBasisStates(N, M)])
    for i in range(len(temp_basis)):
        if(np.prod(temp_basis[i])==1):
            idx = i
            break
    initialStateVec = np.zeros(LENGTH)
    initialStateVec[idx] = 1
    state = StateObj(initialStateVec, None, 'state')

    return state

def getPlot(initDict, tArr):
    """
    Given dictionary of initial parameters and an array of times to evaluate
    at, calculate the Renyi entropy and plot.
    """
    entropy_arr = []
    for t in tArr:
        print('Time: ' + str(t), flush=True)
        tEntropy = initDict['initState'].tevolve(initDict['ham'], t).entropy(
            initDict['N'], initDict['M'], initDict['basis'], initDict['ASIZE'])
        entropy_arr.append(tEntropy)
    plt.plot(tArr, entropy_arr)
    plt.xlabel('Time (s)')
    plt.ylabel('Renyi entropy: $S_{A}$')
    plt.title('Renyi entropy for Bose-Hubbard model, equal bipartition')
    plt.ylim((-0.1, 4))
    #plt.savefig('plot.png', format='png', dpi=200)
    plt.show()
    return
```

```python
def init():
    """
    Initialisation. Gets parameters from user and construct Hamiltonian matrix.
    Returns dict of params.
    """
    N, M, J, U = [float(x) for x in input(
            'Enter params (comma separated: "N, M, J, U"): ').split(', ')]
    N, M = int(N), int(M)
    initialState = getInitialState(N, M)
    hamMatrix, basis = getHamMatrix(N, M, J, U)
    if(bool(int(input('Print 4 lowest energies? (yes:1, no:0): ')))):
        vals, vecs = sp.linalg.eigh(hamMatrix)
        print(vals[:4])
    ASIZE = int(input('Enter size of A subsystem: '))
    return {'N': N, 'M': M, 'J': J, 'U': U, 'initState': initialState,
            'ham': hamMatrix, 'basis': basis, 'ASIZE': ASIZE}

if(__name__ == '__main__'):
    print('In module.')
    initDict = init()
    getPlot(initDict, np.linspace(0, 14, 401))
```