Decoherence of nonclassical state of a harmonic oscillator

Lorenzo Maria Perrone[1, *]
1 Department of Physics, EPFL
*** E-mail: lorenzo.perrone@epfl.ch**

1 QuTip package for Python

QuTIp is a very powerful tool to simulate the evolution of quantum mechanical system. In particular, the convention used in its function of time evolution (ODESOLVE, MESOLVE etc) requires a hamiltonian of the system (for example our harmonic oscillator) plus the definition of jump/collapse operators in the Lindblad form. This is particularly easy to achieve, since such operators can be easily defined (check in the code).
Our formulation of the code is somewhat more general: the temperature of the bath can be changed as a parameter (consequently γ_2 can become relevant) and the ladder/number operators can be extended in a tensor product of many Hilbert spaces: this could be used to simulate the coupling of two or more harmonic oscillator between themselves and with the bath.

```
#import all the necessary packages
import matplotlib.pyplot as plt
from numpy import *
import qutip.settings
from qutip import *
from matplotlib import rcParams
rcParams['font.family'] = 'STIXGeneral'
rcParams['mathtext.fontset'] = 'stix'
rcParams['font.size'] = '14'


#setting up of the parameters

N = 4                 # number of cavity states from 0 to 3
omega0 = 2* pi       # frequency of the harmonic oscillator
gamma1 = 0.2                # energy relaxation rate of oscillator
gamma2 = 0.05                 # dephasing rate of oscillator
n_th = 0.0                  # bath temperature

## define ladder operators and number operator for one oscillator system
```

```
a  = tensor(destroy(N), qeye(1))
number  = tensor(num(N), qeye(1))

## define Hamiltonian and initial state ##
H0 = omega0 * a.dag() * a  #h bar = 1

# tensor product of bath state (ground state)
and initial state of the oscillator NOT NEEDED IN THE COMPUTATION
psi0 = tensor(fock(N,0), (fock(N,0) + fock(N,3))/sqrt(2))
# reduced density matrix for the oscillator at instant t=0
rho_0 = 0.5*fock_dm(N, 0) + 0.5*fock_dm(N, 3)

#sm = tensor(qeye(N), destroy(2))
#sz = tensor(qeye(N), sigmaz())
#H  = omega0 * a.dag() * a + 0.5 * epsilon * sz
#+ g * (a.dag() * sm + a * sm.dag())

## define Lindblad jump (collapse) operators
c_ops = []
c_ops.append(sqrt(gamma1*(1+n_th)) * a)      #L1-
c_ops.append(sqrt(gamma1*n_th) * a.dag())    #L1+
c_ops.append(sqrt(gamma2) * number)          #L2

## Evolution of the system ##
tlist = linspace(0, 20, 50)
#rho_t = odesolve(H, psi0, tlist, c_ops) not sure
rho_t = odesolve(H0, rho_0, tlist, c_ops, [])


#Plot results

#this is useful if one wants to plot the wigner functions
#fig, axes = plt.subplots(2,1, figsize=(12,3))
#plot_wigner_fock_distribution(rho_0, fig= fig, ax=axes[0], cmap=cm.RdBu, title="i
#plot_wigner_fock_distribution(rho_t(100), fig= fig, ax=axes[1], title="final state
#fig.tight_layout()
#plt.show()

rho11 = [];
rho00 = [];
rho22 = [];
rho33 = [];
for i in range(50):
    rho00.append(rho_t[i].data[0,0]);
    rho11.append(rho_t[i].data[1,1]);
    rho22.append(rho_t[i].data[2,2]);
```

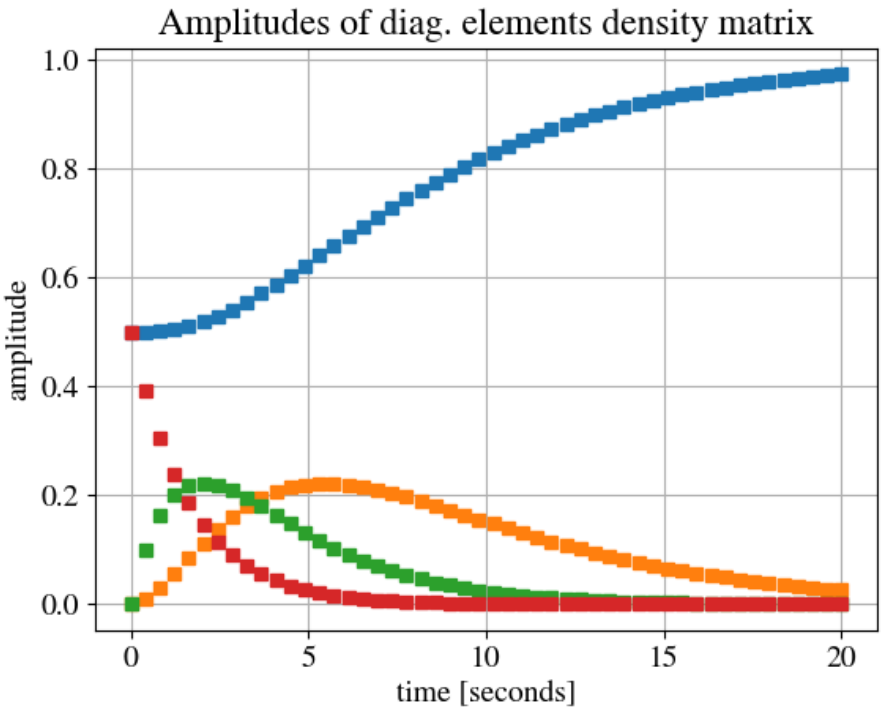

Figure 1. Simulation of the evolution of the diagonal terms of the density matrix. The parameter γ_1 and the elapsing time must of course be tuned with each other.

```
    rho33.append(rho_t[i].data[3,3]);
fig = plt.plot()
plt.plot(tlist, rho00, 's')
plt.plot(tlist, rho11, 's')
plt.plot(tlist, rho22, 's')
plt.plot(tlist, rho33, 's')
plt.title("Amplitudes of diag. elements density matrix")
plt.xlabel("time [seconds]")
plt.ylabel("amplitude")
plt.grid('on')
plt.show()
```

The code can be accessed at HTTPS://GITHUB.COM/LORENZOLMP/STATPHYS4.