Quantum Optics Individual Project - Simulating Atoms in a Cavity

Weijun Yuan

(Dated: May 3, 2022)

Abstract

This project discusses the simulation of an open quantum system consists of atoms and a cavity. We first numerically solve the master equation of a single atom in a cavity and then use quantum jump method to study the same problems. Then, we generalize our discussion to multiple atoms in a cavity.

I. SINGLE ATOM IN A CAVITY

We consider a single atom in a cavity. The atom is initially excited and it is on resonance with the cavity. There are two decay channels of the excited state. The decay rate to free space is Γ_0 and the decay rate to the cavity is Γ_c. (The detail description can be found in the project document on coursework). The Hamiltonian of the system is given by the master equation:

$$\dot{\rho} = \frac{\Gamma_0 + \Gamma_c}{2}(2\sigma_-\rho\sigma_+ - \sigma_+\sigma_-\rho - \rho\sigma_+\sigma_-) \tag{1}$$

A. Solving the master equation numerically

I solve the master equation using the fourth order Runge-Kutta method. The result is shown in Fig. 1. In the plot, we express the time as $\Gamma_0 t$ and use $\Gamma_c/\Gamma_0 = \alpha = 5$. The numerical result agrees with the analytical result very well. We solve the equation analytically as the following

$$\langle e|\,\dot{\rho}\,|e\rangle = \frac{\Gamma_0 + \Gamma_c}{2}\,\langle e|\,(2\sigma_-\rho\sigma_+ - \sigma_+\sigma_-\rho - \rho\sigma_+\sigma_-)\,|e\rangle \tag{2}$$

The first term on the right-hand side is zero as $\sigma_+\,|e\rangle = 0$ and the last two terms give:

$$\dot{\rho}_{ee} = -(\Gamma_0 + \Gamma_c)\rho_{ee} \tag{3}$$

Given the initial condition $\rho_{ee}(0) = 1$, we have the analytical solution as:

$$\rho_{ee}(t) = e^{-(\Gamma_0+\Gamma_c)t} \tag{4}$$

B. Quantum jump method

For this problem, there is only one quantum jump operator, which is given by $c = \sqrt{\Gamma_0 + \Gamma_c}\sigma_-$. The effective Hamiltonian reads

$$H_{eff} = -\frac{i}{2}c^+c \tag{5}$$

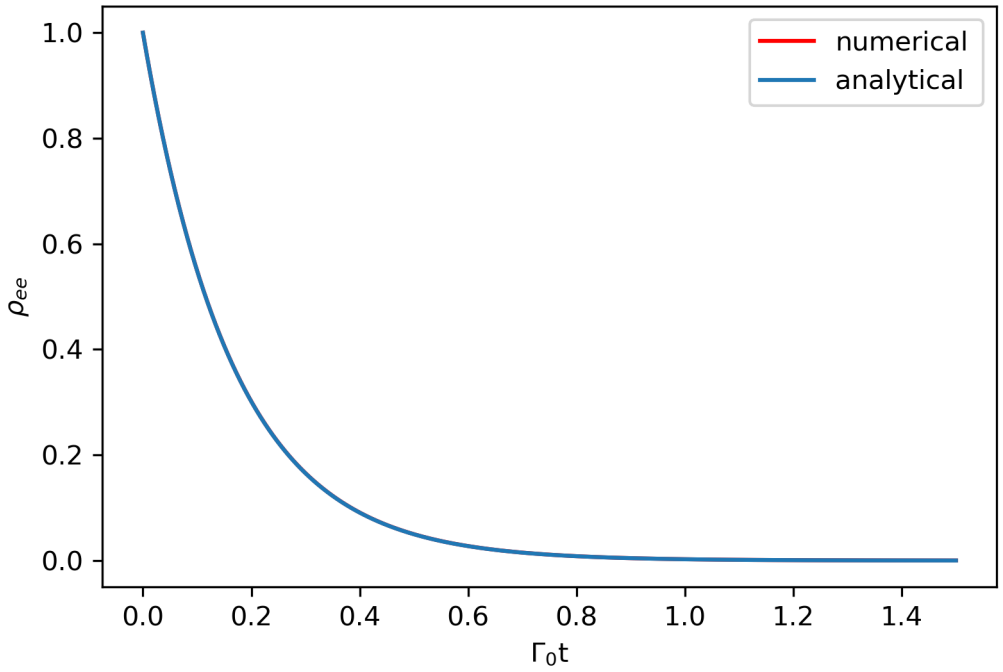


FIG. 1: numerical solution of the master equation. The numerical and analytical results are plotted together.

Then, the master equation can be written as

$$\dot{\rho} = -i(H_{eff}\rho - \rho H_{eff}^{+}) + c^{+}\rho c \tag{6}$$

Then, we can use the first order Monte Carlo wavefunction method described in Daley's paper to calculate the quantum trajectories. Fig. 2 shows 50 quantum jump trajectories. $\delta t = 0.0025$ for each time step. We can see that the trajectories is slightly above 1 when the time is larger, I believe it is due to the normalization issue when applying the first order approximation on the unitary evolution. This issue will be mitigated by taking the average over many trajectories. In Fig. 3, we showed that the quantum jump method converges to the analytical solution as expected when the number of quantum trajectories increases.

II. N ATOMS IN A CAVITY

We extend our previous discussion to N atoms in a cavity. They are initially excited, so the initial state is $|\psi(t=0)\rangle = |e\rangle^{\otimes N}$. The master equation is then given by

$$\dot{\rho} = \frac{1}{2}\sum_{i,j=1}^{N}(\Gamma_c \cos k_c x_i \cos k_c x_j + \Gamma_0 \delta_{ij}))(2\sigma_-^j \rho \sigma_+^i - \sigma_+^i \sigma_-^j \rho - \rho \sigma_+^i \sigma_-^j) \tag{7}$$

where k_c is the cavity mode wave-vector and x_i are the atomic positions. If the atoms are put in the antinode of the cavity, $\cos k_c x_i = 1$ for all i.

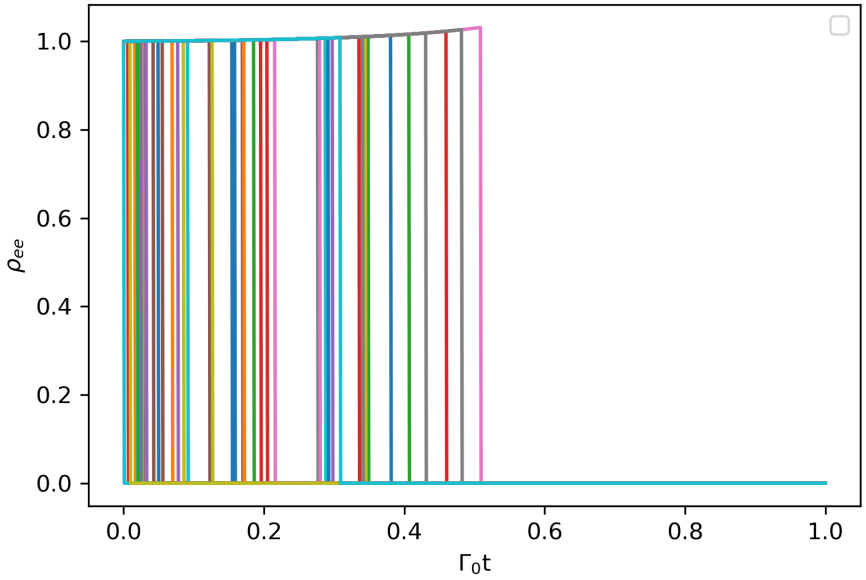
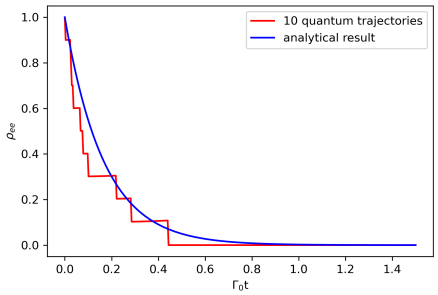


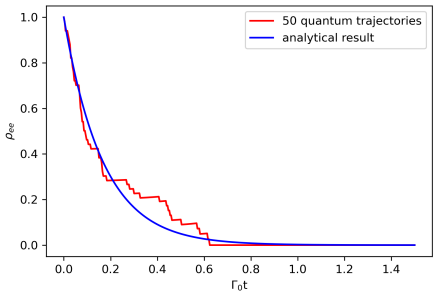
FIG. 2: 50 quantum jump trajectories. The improper normalization of some trajectories at longer time may be caused by the first order approximation of the unitary evolution of wavefunction. The average of those quantum trajectories gives the result shown in Fig 3b.

A. Solving the master equation numerically

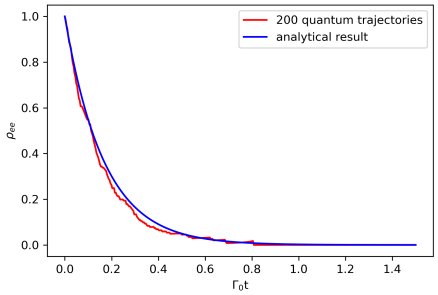
Fig. 4 shows the population in the excited state $\langle \sum_i \sigma_{ee}^i \rangle$. Using the numerical method to solve the master equation, my computer can handle up to 8 atoms. Here I plot the time required to solve the master equation as a function of the number of atoms in Fig. 5. More than 20 minutes is required to solve atom number N = 8 and the time required scale exponentially. It shows that going beyond 8 atoms using numerical method is getting demanding to my computer.



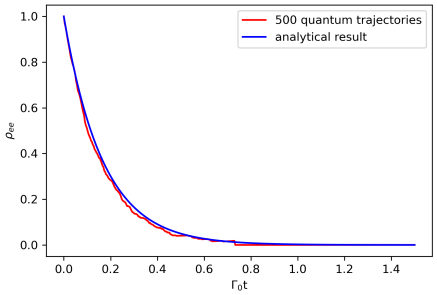
(a) 10 quantum trajectories



(b) 50 quantum trajectories



(c) 200 quantum trajectories



(d) 500 quantum jump trajectories

FIG. 3: Average of different quantum trajectories compared with the analytical solution. When the number of quantum jumps increases, the result converges to the analytical solution.

B. Rate of change of excited population

The rate of change in the excited state population is calculated by choosing $\Gamma_0 = 0$. This quantity is related to the photon emission rate into the cavity R by,

$$R = -\frac{d\langle \sum_i \sigma_{ee}^i(t) \rangle}{dt} \tag{8}$$

The result of the photon emission rate over time is plotted in Fig. 6. The time is dimensionless as it is scaled to $\Gamma_c t$ in this plot. We see that the rate is non-monotonic in time, and there is a peak. This is called "Dicke" superradiant burst. From the plot, we can see that the peak increases with atom number N. I define the visibility of the superradiant burst as vis $= R_{max}/R(t=0)$. If vis $= 1$, there is no superradiant. If it is larger than 1, there is super-radiant. The larger the visibility, the stronger the super-radiant effect is. The scaling of the visibility is plotted in Fig. 7. The green solid line is the exponential fit, which shows

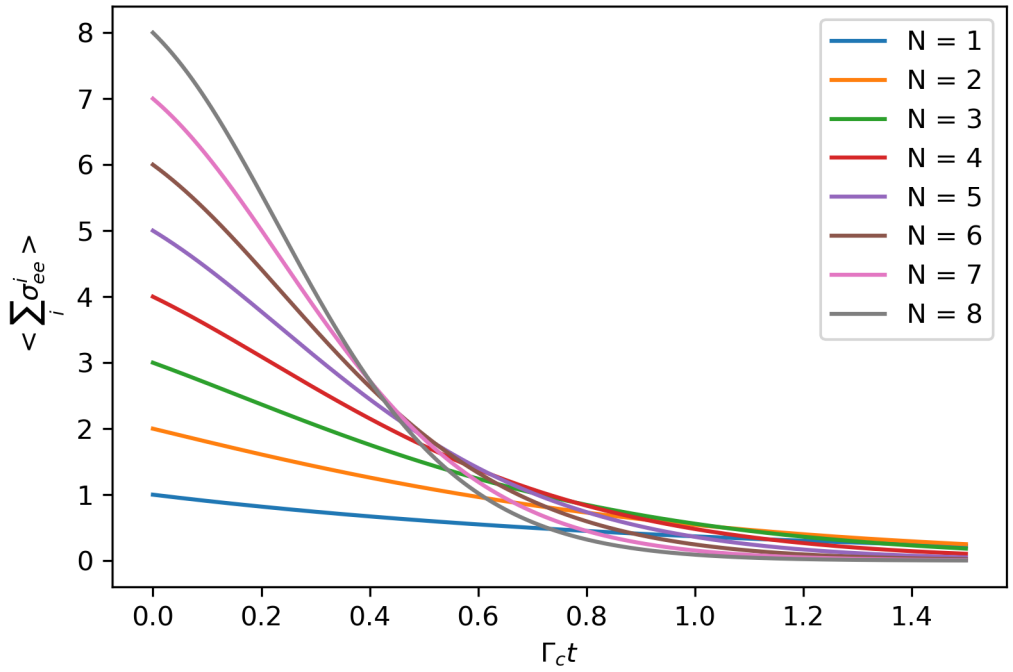


FIG. 4: The population in the excited state as a function of time for different N values. The results are obtained by solving the master equation numerically

that the visibility scales exponentially with the atom number N.

C. Effect of Γ_0

To study the effect of Γ_0, we fix the atom number N to be 5. Fig 8 shows that when Γ_0 is approaching Γ_c from zero, the effect of super-radiance is less visible. From Fig 9, we observe that the vsibility decays exponentially with Γ_0. This phenomenon can be understood as the follows. The super-radiant burst is a many-body effect due to the interaction between atoms. The interaction of atoms are mediated by the photons in the cavity. When Γ_0 is zero, there is no photon leaking to the free space, so the coupling between atoms are the strongest. When Γ_0 is getting larger, the atoms emit more photons to the free space, so the coupling between atoms are weaker, then the super-radiant effect is weaker as well.

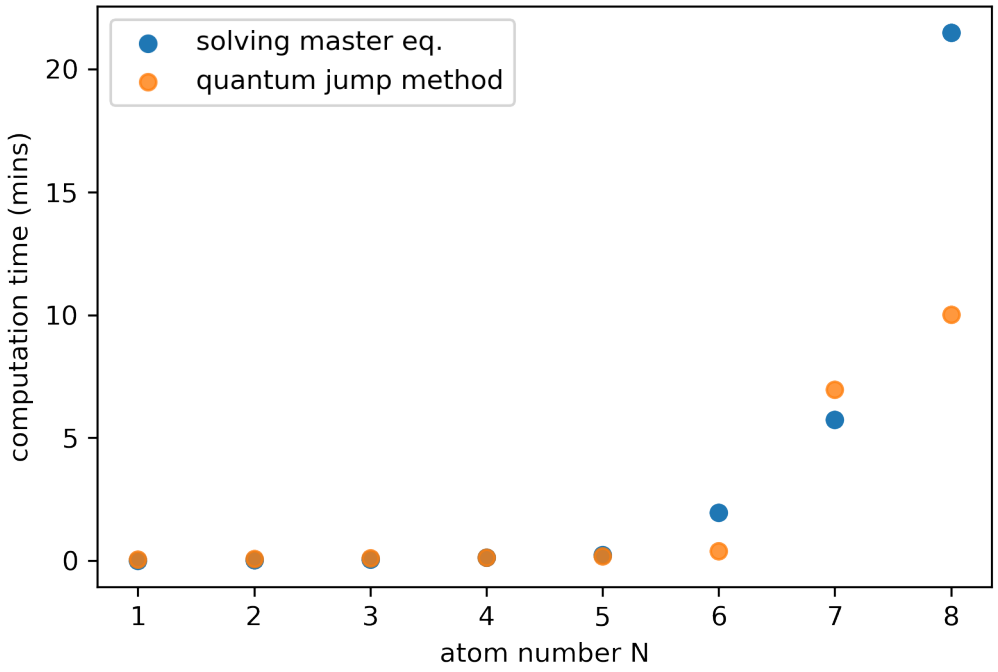


FIG. 5: The computational time required for solving master equation and using the quantum trajectories. 500 times steps are used for this comparison

D. Quantum jump method

For multi-atom system, the challenge of quantum jump method is to find the correct quantum jump operator. First we need to diagonalize the Γ_{ij} matrix to obtain the eigenvalues $\{\Gamma_i\}$ and the corresponding eigenvector $\{v_i\}$. Then, we disentangle different degrees of freedom and write down the following operator

$$O_l^+ = \sqrt{\Gamma_l} \sum_{i=1}^{N} (v_l)_i \sigma_+^i \tag{9}$$

The quantum jump operators can be found by taking the Hermitian conjugate of O_l^+. With these new operators, the master equation can rewritten as

$$\dot{\rho} = \frac{1}{2} \sum_{l=1}^{N} (2O_l \rho O_l^+ - O_l^+ O_l \rho - \rho O_l^+ O_l) \tag{10}$$

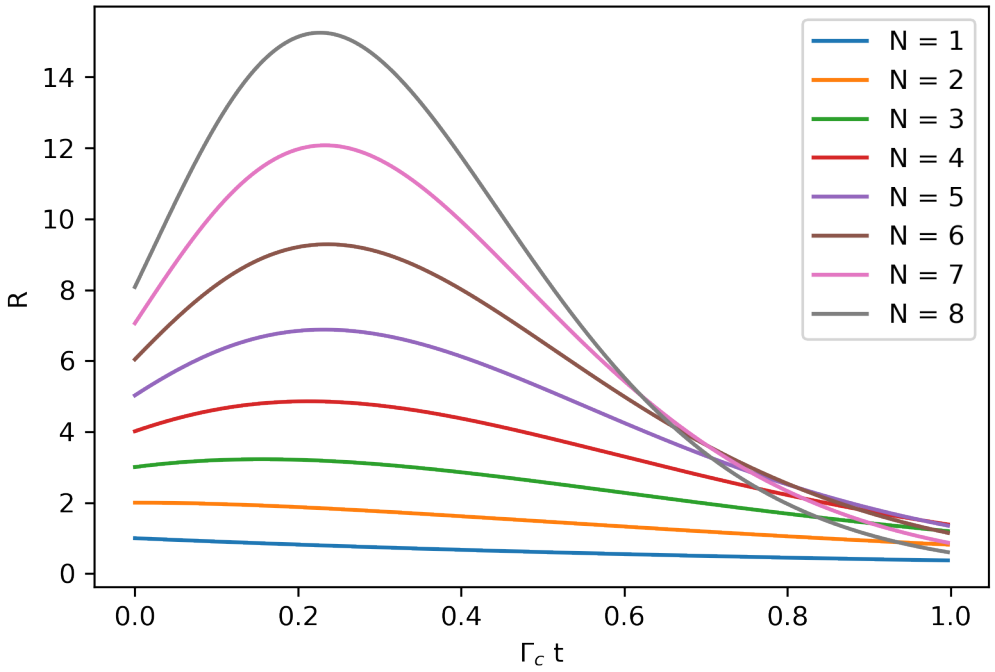


FIG. 6: The rate of photon emission in cavity as a function of time. We observe the super-radiant burst

Then, we can apply the quantum trajectories by defining the effective Hamiltonian H_{eff} as

$$H_{eff} = \frac{i}{2}\sum_{l=1}^{N} O_l^{+} O_l \tag{11}$$

The first order Monte Carlo method is used for the quantum jump trajectories and the resuls are summarized in Fig. 10. The time steps for each quantum trajectory is 500,corresponding to $\delta t = 0.003$, and 500 quantum trajectories are used for each N to achieve a better convergence. $Gamma_0$ is set to $0.2\Gamma_c$. It indicates that the quantum jump method agrees very well with the numerical solution of master equation. Fig. 5 shows how the computational time scales with the number of atoms for both methods. we can see that when the number of atom increase, the quantum jump method become more efficient compared with the numerical solution. For example, when N = 8, the time required for solving the master equation is more than 20 20 minutes, but the time for quantum jump method is slightly larger than 10 minutes. In principle, we can further reduce the number of quantum trajectories and the time steps used to reduce the computational time, while it

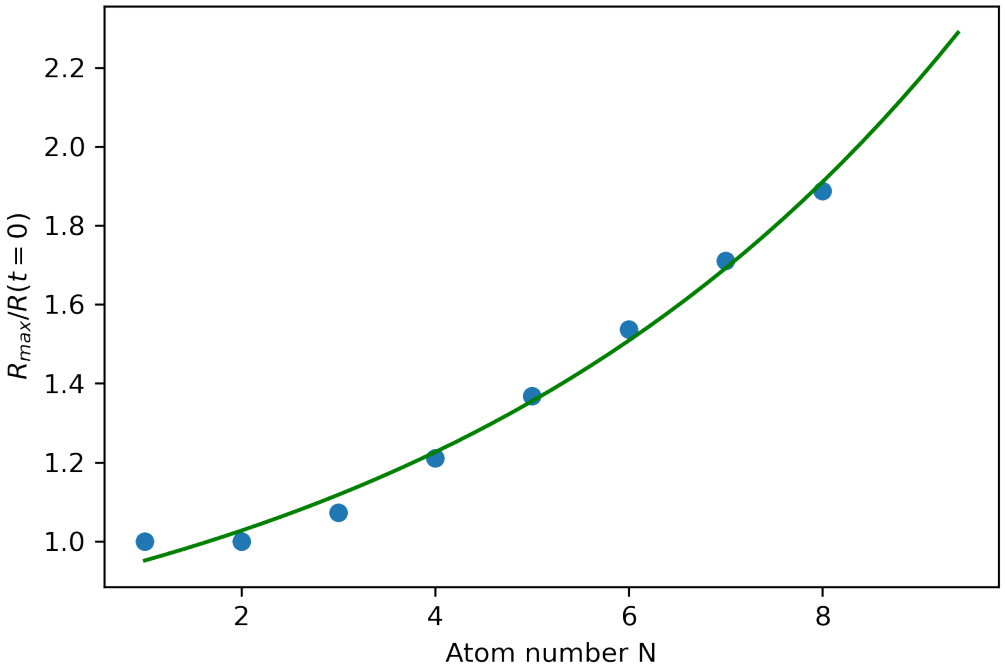


FIG. 7: The visibility as a function of atom number N. Visibility is defined as $R_{max}/R(t=0)$ We observe the exponential scaling between the peak and the atom number N.

may increase the error.

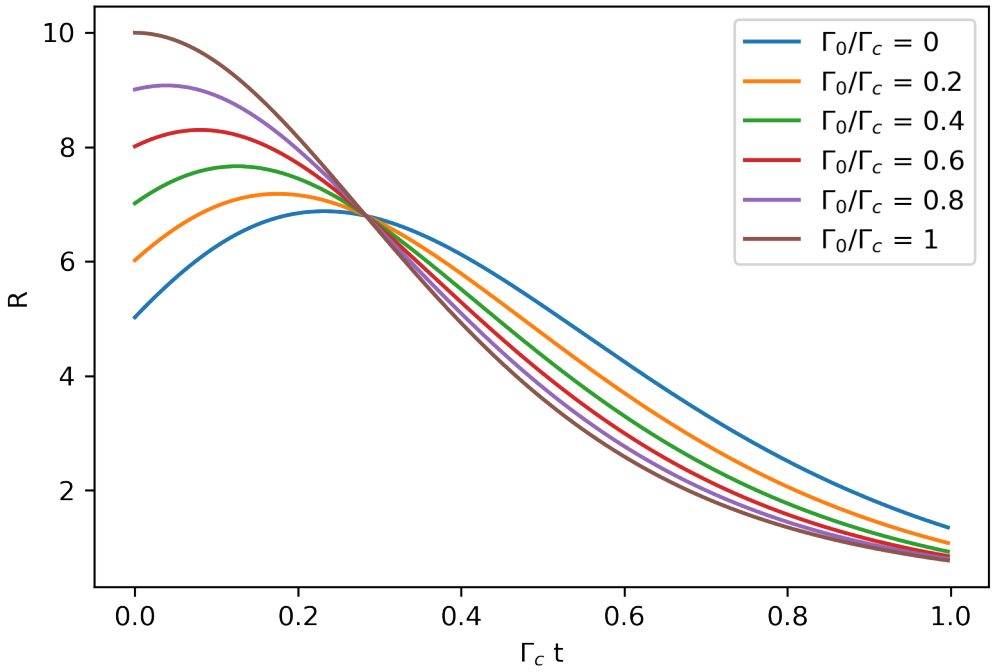


FIG. 8: The rate of photon emission with different Γ_0. The peak gets higher for larger Γ_0

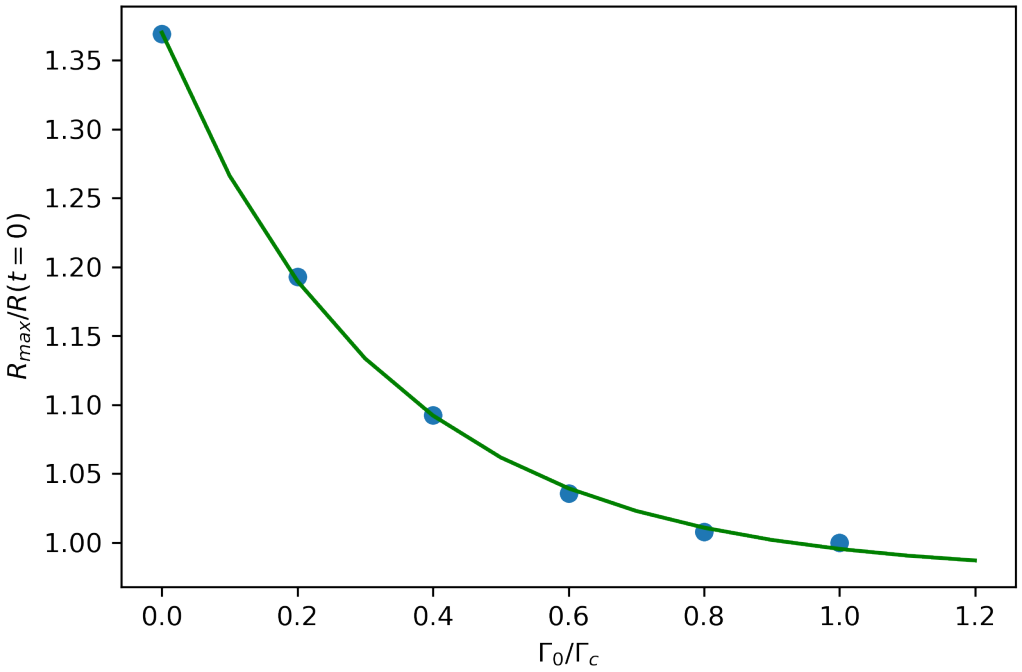
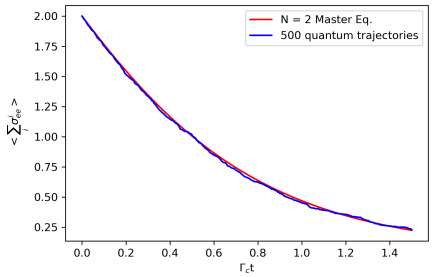
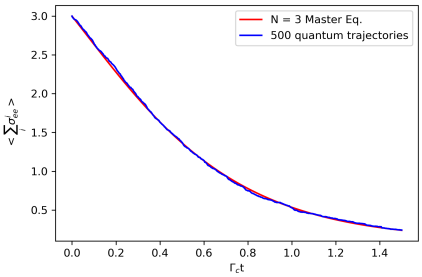


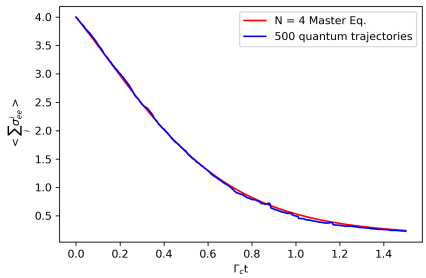
FIG. 9: The visibility as a function of Γ_0. The visibility $R_{max}/R(t=0)$ decays exponentially with Γ_0. The green solid line is the exponential decay fit .



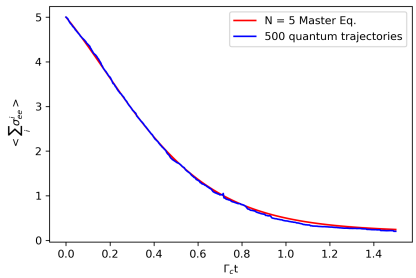
(a) The result of quantum jump method for two atoms



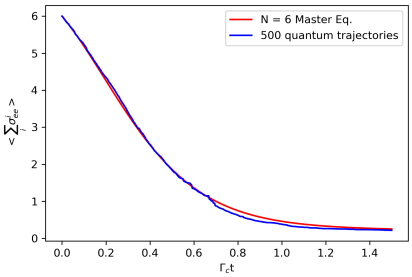
(b) The result of quantum jump method for three atoms



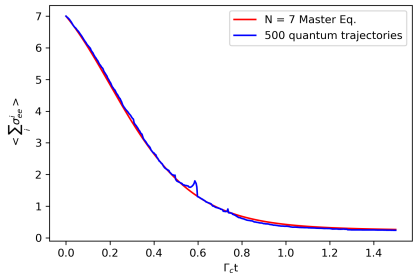
(c) The result of quantum jump method for four atoms



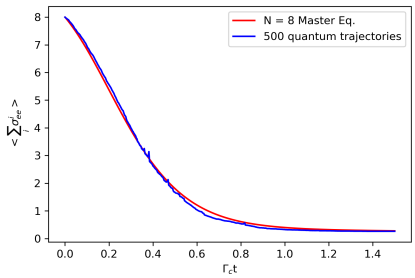
(d) The result of quantum jump method for five atoms



(e) The result of quantum jump method for six atoms



(f) The result of quantum jump method for seven atoms



(g) The result of quantum jump method for eight atoms

FIG. 10: The result of quantum jump method for different atom number N. The red solid line is the result of solving master equation numerically. The blue curve is the result of 500 quantum trajectories. The quantum jump method agrees very well with the numerical solution of master equation.

III. CONCLUSION

To conclude, we have studied the problem of N atoms in a cavity by solving the master equation numerically. This is compared with the results from quantum jump method. They agree well with each other. We also observe the super-radiant phenomenon.

IV. CODE

A. Main program

```
## The program is for quantum optics class

import matplotlib.pyplot as plt
import numpy as np
from numpy import array,matmul,matrix,dot,transpose,identity,average,kron,zeros,trace
import time
import random
from scipy.signal import lfilter
alpha = 5  # gamma_c/gamma0
def f(x,t):
    return -(1+alpha)*x

def anal_result(t):
    return np.exp(-t*(1+alpha))



sigma_p = array([[0,1],[0,0]])
sigma_m = array([[0,0],[1,0]])
sigma_ee = array([[1,0],[0,0]])

def f_3 (rho,t=0):
    new_mat = 2*matmul(sigma_m,matmul(rho,sigma_p))-matmul(sigma_p,matmul(sigma_m,rho
    fx = new_mat*(1+alpha)/2
```

```
    return fx
# rho = array([[0,0],[1,1]])
# print(f_3(rho))

def q1_matrix():
    #main function of question 1
    a = 0      # starting time
    b = 1.5     # end time
    N = 1000   # time step
    h = (b-a)/N
    t_list = np.linspace(a,b,N)

    rhoee_list = []
    rhoeg_list = []
    rhoge_list = []
    rhogg_list = []
    tem_r = np.array([[1.0,0.0],[0.0,0.0]],dtype = float) # initial condition
    for nt in t_list:
        #multivariable Runge-Kutta method
        ee = tem_r[0,0]
        eg = tem_r[0,1]
        ge = tem_r[1,0]
        gg = tem_r[1,1]
        rhoee_list.append(ee)
        rhoeg_list.append(eg)
        rhoge_list.append(ge)
        rhogg_list.append(gg)

        k1 = h*f_3(tem_r,nt)
        k2 = h*f_3(tem_r+0.5*k1,nt+0.5*h)
        k3 = h*f_3(tem_r+0.5*k2,nt+0.5*h)
        k4 = h*f_3(tem_r+k3,nt+h)
```

```
        tem_r +=(k1+2*k2+2*k3+k4)/6
    plt.plot(t_list,rhoee_list,"r-",label = "numerical")
    #plt.plot(t_list,rhogg_list,"b-",label = "rho_gg")
    # plt.plot(t_list,rhoge_list,"g-",label = "rho_ge")
    # plt.plot(t_list,rhoge_list,"y-",label = "rho_eg")
    plt.legend()
    plt.ylabel(r'$\rho_{ee}$')
    plt.xlabel(r"$\Gamma_{0}$t")
    analytical = anal_result(t_list)
    plt.plot(t_list,analytical,label = "analytical ")
    plt.legend()
    plt.savefig("/Users/weijunyuan/OneDrive - HKUST Connect/paper_book_pdf/courses/qu

#q1_matrix()


##############################################
#Quantum jump for single atom
def delta_p(phi,c_m,c_m_d,h):

    return dot(phi.conj(),matmul(c_m_d,matmul(c_m,phi)))*h
def phi_1(phi_t,action):
    return matmul(action,phi_t)

def quantum_jump():
    a = 0
    b = 1.5
    N = 400
    h = (b-a)/N     ### time interval
    c_m = sigma_m*np.sqrt((1+alpha))
    c_m_d = sigma_p*np.sqrt(1+alpha)
    H_eff = -(1j/2)*matmul(c_m_d,c_m)
```

```
    action = identity(2)- 1j*H_eff*h
    r_1 = 0
    phi_0 = array([1,0])
    phi_e = []
    phi_g = []
    phi = phi_0
    rho_ee = [dot(phi_0[0],phi_0[0])]
    rho_gg = [dot(phi_0[1],phi_0[1])]
    for i in range(1,N):
        r_1 = random.uniform(0,1)
        d_p = delta_p(phi,c_m,c_m_d,h)
        if r_1 > d_p:
            new_phi = phi_1(phi,action)/np.sqrt(1-d_p)
        else:
            new_phi = matmul(c_m,phi)/np.sqrt(d_p/h)
        #print(new_phi)
        phi_e.append(new_phi[0])
        phi_g.append(new_phi[1])
        rho_ee.append(dot(new_phi[0],new_phi[0]).real)
        phi = new_phi
    return rho_ee
#quantum_jump()



def single_atom_qj_solver():
    a = 0
    b = 1.5
    N = 400
    t_list = np.linspace(a,b,N)
    rho_ee_list = []
    for i in range(1):
```

```
        rho_ee = quantum_jump()
        rho_ee_list.append(rho_ee)
    rho_ee_list = array(rho_ee_list)
    t_list = np.linspace(a,b,N)
    rho_ee_avg = average(rho_ee_list,axis = 0)
    plt.plot(t_list,rho_ee_avg,"r-",label = "1 quantum trajectories")
    #analytical = anal_result(t_list)
    #plt.plot(t_list,analytical,'b-',label = "analytical result")
    plt.legend()
    plt.ylabel(r'$\rho_{ee}$')
    plt.xlabel(r"$\Gamma_{0}$t")
    plt.savefig("/Users/weijunyuan/OneDrive - HKUST Connect/paper_book_pdf/courses/qu
#single_atom_qj_solver()

################ multiatoms

def initial_state(N):
    phi_0 = array([1.,0.],dtype = "complex128")
    phi_N = phi_0
    for i in range(N):
        phi_N = kron(phi_N,phi_0)
    return phi_N



def I_N(N):
    return identity(2**N)

def Sigma_m(N,j):
    return kron(I_N(j-1),kron(sigma_m,I_N(N-j)))

def Sigma_ee(N,j):
    return kron(I_N(j-1),kron(sigma_ee,I_N(N-j)))
```

```
def Sigma_p(N,j):
    return kron(I_N(j-1),kron(sigma_p,I_N(N-j)))
def rho0_N(rho_0,N):
    rho_N = identity(1)
    for i in range(N):
        rho_N = kron(rho_N,rho_0)
    return rho_N
gamma_c = 1
gamma_0 = 0.2
def H_N(rho,N,Gamma_0,t=0):
    H_jj = zeros((2**N,2**N))
    H_ij = zeros((2**N,2**N))
    Rho = rho ### initial condition
    for i in range(1,N+1):
        for j in range(1,N+1):
            if i == j:
                H_jj += 2*matmul(Sigma_m(N, j),matmul(Rho,Sigma_p(N, j)))- \
                        matmul(Sigma_p(N, j),matmul(Sigma_m(N, j),Rho))- \
                        matmul(Rho,matmul(Sigma_p(N, j),Sigma_m(N, j)))
            else:
                H_ij += 2*matmul(Sigma_m(N, j),matmul(Rho,Sigma_p(N, i)))- \
                        matmul(Sigma_p(N, i),matmul(Sigma_m(N, j),Rho))- \
                        matmul(Rho,matmul(Sigma_p(N, i),Sigma_m(N, j)))
    H = H_jj*(gamma_c+Gamma_0)+H_ij*gamma_c
    return H/2
# rhoi = rho0_N(rho_0, 2)
# print(H_N(rhoi,2))
def pop_sum(N):
    ### give the population operator for N atoms
    pop_operator = zeros((2**N,2**N))
    for i in range(1,N+1):
```

```
        pop_operator += Sigma_ee(N, i)
    return pop_operator
def exp_value(rho,operator,N):
    ### find the expectation value of an operator for given rho
    expect = trace(matmul(rho,operator))
    return expect

rho_0 = array([[1,0],[0,0]])
def multiatom_solver(N=1,Gamma_0=0):
    a = 0     # starting time
    b = 1.5  # end time
    n = 500  # time step
    h = (b-a)/n
    t_list = np.linspace(a,b,n)
    Rho_0 = rho0_N(rho_0, N)
    population = pop_sum(N)
    tem_r = Rho_0# initial condition
    pop_list = []
    rate_list = []
    for nt in t_list:
        #multivariable Runge-Kutta method
        pop_exp = exp_value(tem_r, population, N)
        pop_list.append(pop_exp)
        k1 = h*H_N(tem_r,N,Gamma_0,nt)
        k2 = h*H_N(tem_r+0.5*k1,N,Gamma_0,nt+0.5*h)
        k3 = h*H_N(tem_r+0.5*k2,N,Gamma_0,nt+0.5*h)
        k4 = h*H_N(tem_r+k3,N,Gamma_0,nt+h)
        tem_r +=(k1+2*k2+2*k3+k4)/6
    #plt.plot(t_list,pop_list,"r-",label = "N = " + str(N)+" Master Eq. ")
    for i in range(1,n):
        rate = (pop_list[i]-pop_list[i-1])/h
        rate_list.append(rate)
```

```
    rate_list = -1*array(rate_list)

    return pop_list,rate_list


#N is the number of atoms
a = 0     # starting time
b = 1.5  # end time
n = 500  # time step
h = (b-a)/n
t_list = np.linspace(a,b,n)
start = time.time()
N = 8
exp_ee,rate = multiatom_solver(N,gamma_0)
plt.plot(t_list,exp_ee,"r-",label = "N = " + str(N)+" Master Eq. ")
end = time.time()
print(end-start)

'''
a = 0     # starting time
b = 1  # end time
n = 300  # time step
h = (b-a)/n
t_list = np.linspace(a,b,n)
for i in range(1,7):
    exp_ee,rate = multiatom_solver(i,gamma_0)
    plt.plot(t_list,exp_ee,label = "N = "+str(i))
    plt.legend()
    plt.xlabel(r'£\Gamma_c t£')
    plt.ylabel(r'£<\sum_{i} \sigma_{ee}^i>£')
    plt.savefig("/Users/weijunyuan/OneDrive - HKUST Connect/paper_book_pdf/courses/
'''
```

```
def rate_N(N):
    a = 0      # starting time
    b = 1   # end time
    n = 300   # time step
    t_list = np.linspace(a,b,n)
    rate_ll = []
    pop_ll = []
    for i in range(1,N):
        pop_i,rate_i = multiatom_solver(i)
        plt.plot(t_list[0:n-1],rate_i,label = "N = "+str(i))
        rate_ll.append(rate_i)
        pop_ll.append(pop_i)
    plt.ylabel(r"R")
    plt.xlabel(r"$\Gamma_c$ t")
    plt.legend()
    plt.savefig('/Users/weijunyuan/OneDrive - HKUST Connect/paper_book_pdf/courses/qu
    plt.show()
    rate_max = []
    for rate in rate_ll:
        rate_max_i = max(rate)
        rate_max.append(rate_max_i/rate[0])
    n = np.arange(1,N)
    plt.scatter(n,rate_max)
    plt.xlabel("N")
    plt.ylabel(r"$R_{max}$")
    plt.savefig('/Users/weijunyuan/OneDrive - HKUST Connect/paper_book_pdf/courses/qu
    print(rate_max)

#rate_N(N = 9)

def rate_gamma():
    a = 0      # starting time
```

```
    b = 1  # end time
    n = 300  # time step
    t_list = np.linspace(a,b,n)
    rate_ll = []
    pop_ll = []
    Gamma = [0,0.2,0.4,0.6,0.8,1]
    for gamma_0 in Gamma:
        pop_i,rate_i = multiatom_solver(N = 5,Gamma_0 = gamma_0)
        plt.plot(t_list[0:n-1],rate_i,label = r"$\Gamma_{0}/\Gamma_{c}$ = "+str(gamma
        rate_ll.append(rate_i)
        pop_ll.append(pop_i)
    plt.ylabel(r"R")
    plt.xlabel(r"$\Gamma_c$ t")
    plt.legend()
    plt.savefig('/Users/weijunyuan/OneDrive - HKUST Connect/paper_book_pdf/courses/qu
    plt.show()
    rate_max = []
    for rate in rate_ll:
        rate_max_i = max(rate)
        rate_max.append(rate_max_i/rate[0])
    plt.scatter(Gamma,rate_max)
    plt.xlabel(r"$\Gamma_0/\Gamma_c$")
    plt.ylabel(r"$R_{max}$")
    plt.savefig('/Users/weijunyuan/OneDrive - HKUST Connect/paper_book_pdf/courses/qu
    print(rate_max)
#rate_gamma()

########==========multiatom quantum jump=========##############

def gamma_matrix(N,Gamma_0,Gamma_c):
    ###define the gamma matrix
    Gamma = identity(N)*(Gamma_0)+np.ones((N,N))*Gamma_c
```

```
    return Gamma


def collective_jump_O(N,w,v):
    O_list = []
    O_d_list = []
    sigma_p_list = []
    sigma_m_list = []
    for i in range(1,N+1):
        sigma_p_list.append(Sigma_p(N,i))
        sigma_m_list.append(Sigma_m(N,i))
    for l in range(N):
        O_d_l = zeros((2**N,2**N))
        for i in range(0,N):
            O_d_l = O_d_l + v[:,l][i]*sigma_p_list[i]
        O_d_list.append(O_d_l*np.sqrt(w[l]))
        #O_list.append(np.sqrt(w[l])*np.transpose(np.conjugate(O_d_l)))
    for l in range(N):
        O_l = zeros((2**N,2**N))
        for i in range(0,N):
            O_l = O_l + np.conjugate(v[:,l][i])*sigma_m_list[i]
        O_list.append(O_l*np.sqrt(w[l]))
    return O_list,O_d_list
def delta_p_N(N,phi,c_m,c_m_d,h):
    d_p_list = []
    d_p = 0
    for i in range(N):
        d_p_m = dot(phi.conj(),matmul(c_m_d[i],matmul(c_m[i],phi)))*h
        d_p_list.append(d_p_m)
        d_p = d_p+d_p_m
    return d_p,d_p_list


def H_eff_gen(N,c_m,c_m_d):
```

```
    H = zeros((2**N,2**N))
    for i in range(N):
        H = H+ matmul(c_m_d[i],c_m[i])
    return -1j/2*H
def phi_0_N(N):
    ### return the initial state fir
    phi_list = [1]
    for i in range(2**N-1):
        phi_list.append(0)
    return np.array(phi_list)
def interval_gen(N,d_p,d_p_list):
    d_p_array = np.array(d_p_list)
    normalized_array = d_p_array/d_p
    interval_b = 0  ### interval upper boundary
    b_list = []   ### boundary list
    for d_p_m in normalized_array:
        interval_b = interval_b+d_p_m
        b_list.append(interval_b.real)
    return b_list
def pop_2(phi,population):
    return dot(phi.conj(),matmul(population,phi))
def quantum_jump_N(N,N_i,h,c_m,c_m_d,action,phi_0,population):
    r_1 = 0
    r_2 = 0
    phi = phi_0
    d_p,d_p_list= delta_p_N(N,phi,c_m,c_m_d,h)
    #rho_ee = [dot(phi_0[0],phi_0[0])]
    #rho_gg = [dot(phi_0[1],phi_0[1])]
    pop_list = []
    for i in range(N_i):
        r_1 = random.uniform(0,1)
        d_p,d_p_list= delta_p_N(N,phi,c_m,c_m_d,h)
```

```
        if r_1 > d_p:
            new_phi = phi_1(phi,action)/np.sqrt(1-d_p)
        else:
            r_2 = random.uniform(0,1)
            interval_list= interval_gen(N,d_p,d_p_list)
            j = 0
            true = True
            while true:
                interval = interval_list[j]
                if r_2 <= interval :
                    new_phi = phi_1(phi,c_m[j])/np.sqrt(d_p_list[j]/h)
                    true = False
                else:
                    j = j+1
        # phi_matrix = np.mat(phi)
        # tem_r = matmul(np.transpose(np.conjugate(phi_matrix)),phi_matrix)   ### de
        #pop_exp = exp_value(tem_r, population, N)
        pop_exp = pop_2(phi,population)
        phi = new_phi
        pop_list.append(pop_exp.real)
    return pop_list


def multiatom_qj_solver(N = 1,n = 100):
    a = 0
    b = 1.5
    N_i = 500
    h = (b-a)/N_i
    t_list = np.linspace(a,b,N_i)
    rho_ee_list = []
    Gamma_N = gamma_matrix(N,gamma_0,gamma_c)
    w,v = np.linalg.eig(Gamma_N)
```

```
for i in range(N):
    if w[i]< 0:
        w[i] = 0
c_m,c_m_d = collective_jump_O(N,w,v)
H_eff = H_eff_gen(N, c_m, c_m_d)
action = identity(2**N)- 1j*H_eff*h
population = pop_sum(N)
phi_0 = phi_0_N(N)  ### the initial state is all atoms get excited
for i in range(n):
    rho_ee = quantum_jump_N(N,N_i,h,c_m,c_m_d,action,phi_0,population)
    rho_ee_list.append(rho_ee)
rho_ee_list = array(rho_ee_list)
t_list = np.linspace(a,b,N_i)
rho_ee_avg = average(rho_ee_list,axis = 0)
plt.plot(t_list,rho_ee_avg,"b-",label = str(n)+" quantum trajectories")
#analytical = anal_result(t_list)
#plt.plot(t_list,analytical,'b-',label = "analytical result")
plt.legend()
plt.ylabel(r'$<\sum_{i} \sigma_{ee}^i>$')
plt.xlabel(r"$\Gamma_{c}$t")
# rate_list = []
# for i in range(1,N_i):
#     rate = (rho_ee_avg[i]-rho_ee_avg[i-1])/h
#     rate_list.append(rate)
# rate_list = -1*array(rate_list)
# n = 15  # the larger n is, the smoother curve will be
# b = [1.0 / n] * n
# a = 1
# y = lfilter(b,a,rate_list)
# plt.plot(t_list[0:N_i-1],y)
plt.savefig("/Users/weijunyuan/OneDrive - HKUST Connect/paper_book_pdf/courses/qu
plt.show()
```

```
#main()
#exp_ee,rate = multiatom_solver(4,gamma_0)
start = time.time()
multiatom_qj_solver(N,n = 500)
end = time.time()
print(end-start)
```

B. fitting

```
import matplotlib.pyplot as plt

import numpy as np
from scipy.optimize import curve_fit



x_data = np.array([1,2,3,4,5,6,7,8,9,10])
y_data = np.array([0.09,0.33,0.77,1.5,2.8,22,66,288,858,19547])



def exp_i(x,A,B,C):
    return A*np.exp(B*x)+C
popt, pcov = curve_fit(f_x,x_data,y_data)
x = np.arange(1,10.5,0.2)
plt.scatter(x_data, y_data/60)
plt.xlabel("atom number")
plt.ylabel("computational time (mins)")
plt.plot(x,exp_i(x,popt[0],popt[1],popt[2])/60)
plt.savefig("/Users/weijunyuan/OneDrive - HKUST Connect/paper_book_pdf/courses/quantu
plt.show()
```

```
x_data = np.arange(1,9)
y_data = np.array([1.0, 1.0, 1.0731646300205278, 1.2103230256940611, 1.36890263700945
popt, pcov = curve_fit(exp_i,x_data,y_data)
x = np.arange(1,9.5,0.2)
plt.scatter(x_data, y_data)
plt.xlabel("Atom number N")
plt.ylabel(r"$R_{max}/R(t=0)$")
plt.plot(x,exp_i(x,popt[0],popt[1],popt[2]),'g-')
print(popt[0],popt[1],popt[2])
plt.savefig("/Users/weijunyuan/OneDrive - HKUST Connect/paper_book_pdf/courses/quantu
plt.show()


def exp_d(x,A,B,C):
    return A*np.exp(-B*x)+C
x_data = np.array([0,0.2,0.4,0.6,0.8,1])
y_data = np.array([1.3689026370094546, 1.192817210064132, 1.0922840868626325, 1.03586
popt, pcov = curve_fit(exp_d,x_data,y_data)
plt.scatter(x_data, y_data)
x = np.arange(0,1.3,0.1)
plt.xlabel(r'$\Gamma_0/\Gamma_c$')
plt.ylabel(r"$R_{max}/R(t=0)$")
plt.plot(x,exp_d(x,popt[0],popt[1],popt[2]),'g-')
print(popt[0],popt[1])
plt.savefig("/Users/weijunyuan/OneDrive - HKUST Connect/paper_book_pdf/courses/quantu
plt.show()

N_data = np.arange(1,8)
t_data = np.array([0.4204280376434326,1.5902106761932373,3.771937847137451,7.46779108
t_2_data = np.array([3.733560085296631,4.990548849105835,6.610157012939453,8.45458793
plt.scatter(N_data,t_data/60)
plt.scatter(N_data,t_2_data/60)
```